RELEASE DESCRIPTION (SRD)
SYSTEM V

unix* system

Western Electric

*Trademark of Bell Laboratories

WECO
301-909

UNIX ™ System

Prospectus

System V

January 1983

© 1983 Western Electric Company, Incorporated

PDP, VAX AND DEC
are trademarks of Digital Equipment Corporation.

UNIX is a trademark of Bell Laboratories

1. EXECUTIVE SUMMARY

UNIX System V is a major release across the following processors:

- DEC+ VAX-11/780
- DEC VAX-11/750
- DEC PDP-11/70

This prospectus presents an overview of the new features provided by System V and the procedures required for transition to the new system.

The prospectus is divided into five sections.

1. Executive Summary
2. System V Overview
3. Features: A description of the new features supplied in this release.
4. Transition to System V: A synopsis of the changes that must be taken into account when moving to the new system.
5. Items no longer supported.

2. SYSTEM OVERVIEW

This section contains a listing of all the new features in System V. For a more detailed description see Section 3.

2.1 Supported Processors

System V will be issued and supported on the following processors:

VAX-11/780
VAX-11/750
PDP-11/70

The PDP system has reduced system capabilities. The features that have limited availability on the supported processors are identified in the following sections.

2.2 System V Features

The following section is a brief feature overview and list of individual features. A detailed description of the features is given in Section 3.

2.2.1 Kernel Enhancements and Changes.

2.2.1.1 Operating System Performance. The performance of the operating system on the VAX systems will be improved by a larger block file system, eliminating redundant work and providing a larger number of I/O buffers. The new features are:

- 1 Kbyte block file systems.
- Physical I/O buffer facility.
- Larger buffer pool.
- File table free list.
- Faster *fork* system call.
- Faster *open* system call (inode hashing).
- Hashing of internal sleep/wakeup mechanism (also on PDP systems).

2.2.1.2 File System Changes. The file system was enhanced to maintain the consistency of the disk information. This will eliminate a class of disk errors associated with system crashes. The new features are:

- Ordered writes.
- Dual *fsck* capability.
- Faster *fsck* option.
- Improved file-system access.
- *Fuser* command.
- System activity report.

2.2.1.3. Operating System: Common Features and Enhancements. Among the new features System V provides are an Interprocess Communication interface and *init* and *getty*. The complete list is:

- Processor type identification.
- Interprocess communication.
- *Init* and *getty*.
- New general disk driver.

2.2.2 Enhancements and Changes for DEC Processors. The following new devices are supported:

- Large capacity disk (516 Mbyte) drive (VAX only).
- High speed (6250 bpi) tape drive (VAX-11/780 only).
- Medium capacity (124 Mbyte) disk drive (VAX only).
- Medium speed (1600 bpi) tape drive (VAX-11/750 only).
- MOS memory.

2.2.3 Languages and Libraries. The language tools of System V have been changed to provide run-time efficiency and improved performance. The C library in particular has been changed to take advantage of the improvements in the operating system and to perform its functions more efficiently. The list of features includes:

- Software generation system performance improvements (VAX only).
- Improved machine language processors (VAX only).
- Library performance enhancements.
- Profiled versions of the libraries.
- Math library enhancements.
- New library functions.

2.2.4 Communication and Networking. The Communications and Networking Facilities of System V will provide improvements in the currently supported protocols. The list includes:

- Common Synchronous Interface.
- *Uucp* improvements.

2.2.5 Commands and Administration Aids. UNIX System V will provide the user and administrator with the following features:

- Incremental tape back up.
- *vi*[1] screen editor support.
- Accounting enhancements.
- Line printer spooling system.

2.3 Transition Aids

Customers will be required to perform some conversion functions when installing System V (see Sections 4 for more details). Conversion tools and instructions will be supplied for customers currently using Release System III.

The aids available are:

- *Converting to a 1K-block File System*
- *Changes to the UNIX System Archive Format*
- *User Level Changes to the C Language*
- *Common Object File Format*
- *UNIX Operating System Environment Changes: System III to System V*

The following conversion tools will be provided:

- *arcv* (1)—convert archive files from PDP-11 to common archive format
- *fsba* (1)—file system block analyzer

3. NEW FEATURES

3.1 Kernel Enhancements and Changes

Except as noted these changes are only on the VAX systems.

3.1.1 Operating System Performance. The performance of the operating system is improved by increasing the block size of the file system, by increasing the size of the internal system buffers, by eliminating redundant work and providing a larger buffer pool.

3.1.1.1 1 Kbyte Block File Systems. System V supports file systems with 1024 byte blocks and 1024 byte buffers on the VAX systems.[2] This provides for performance improvements I/O bound processes. While the performance increase is dependent on individual system loads, operations that do a large number of reads and writes show dramatic improvement. Performance with 512 byte block file systems will remain the same. (See also Languages and Libraries, Section 3.3.)

3.1.1.2 Physical I/O Buffer Facility. System V supports multiple physical I/O buffers. Once a physical I/O request is scheduled, the requesting program cannot be swapped out. Therefore, it is recommended that the *physbuf* parameter (see *config* (1M)) be set between three (on an 11/70 with <= 1MB of memory) and five (on a VAX with >= 2MB). Note that the faster file system checking feature, *dfsck*, described in 3.1.2.2 requires two or more physical I/O buffers.

---

1. The *vi* screen editor is based on software developed by The University of California, Berkeley, California Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

2. This features achieves the performance of the 1 Kbyte block file system of The University of California, Berkeley 4.1 BSD system while remaining compatible with the "old" 512 byte block file system.

3.1.1.3 Larger Buffer Pool. System V supports a larger number of buffers that can increase the potential for cache "hits" and thus reduce some of the disk I/O traffic. Limitations of the memory management routines previously precluded this increase. The documentation reflects these changes to the system configuration.

3.1.1.4 File Table Free List. A list of free positions in the file table reduces the overhead needed to insert a new entry into this table. The free list eliminates the linear search past all the files opened by dedicated applications initiated at boot time.

3.1.1.5 Faster Fork System Call. The previous two complete passes through the process table to complete a *fork* system call are reduced to one pass. This eliminates some of the overhead of every *fork* system call.

3.1.1.6 Fast Open System Call. Inode table hashing is implemented in System V to improve the performance of the *open* system call. The *open* system call, and all other searches of the in-core inode table, is speeded up by hashing the table and avoiding a linear search. A preliminary version of System V on the VAX showed that CPU time spent in locating an inode is reduced from 9% to 1% of the total CPU time with the addition of inode hashing.

3.1.1.7 Hashing of Sleep/Wakeup Mechanism. All processes waiting upon an event are now kept in a hash table instead of a linear list. This reduces the amount of time that the system spends looking for sleeping processes. (This feature is also in the PDP-11/70 system.)

3.1.2 File System Changes. The following file system changes are only on the VAX systems.

3.1.2.1 Ordered writes. The file system has been made more robust by controlling the order of blocks being written. This enhancement protects the file system from a serious class of errors in the event of a system crash. This scheme orders the writing of the directory and inode information, at link and unlink time, so that the link count is always positive. When a file is linked, the inode is forced to be written out to the disk so the inode information will appear before the directory information. When the file is unlinked, the directory is forced to be written out to the disk so the directory information will appear before the inode information.

3.1.2.2 Dual fsck capability. Dfsck is a new utility that speeds up file system checking on multi-drive machines by executing two *fsck*s in parallel. To do this, the *dfsck* procedure uses the multiple physical I/O buffer feature mentioned in 3.1.1.1. Careful use of this utility can save over 40% on file system checking time.

Due to the file system load balancing required for dual checking, the *dfsck* command should always be executed through the *checkall* (1M) shell procedure.

3.1.2.3 Faster Fsck Option. Since the file system will be more resistant to corruption (see Ordered Writes above), there is an addition to *fsck* to allow for quick recovery from crashes. The addition of a "fast" option (*–f*) to *fsck* will skip phases 2, 3 and 4 (see *fsck* (1M)), which are often the most time-consuming and I/O-intensive phases. It will check the i-list, check and rebuild the free list if necessary, and, then, ensure that there is no damage that could further corrupt the file system. A full *fsck* is still needed at file-save time to assure consistency of the file system.

3.1.2.4 Improved File-System Access. System V provides a new file-system reorganization utility, *dcopy*, which reorders files and directories to optimize file-system access. On an otherwise idle system, a reorganized file system has almost twice the I/O throughput of a randomly organized file system. This applies, for example, to file copying, *finds*, and *fscks*. Note that during reorganization, the file system must be unmounted. See *dcopy* (1M) for details.

3.1.2.5 Fuser. The *fuser* (1M) command allows administrators to identify and kill processes associated with a particular file system, terminal line, or any other file. See *fuser* (1M) for details.

3.1.2.6 System Activity Report (SAR). New performance measurement counters have been added in the operating system for use by the *sar* (1) commands.

3.1.3 Operating System: Common Features and Enhancements.

3.1.3.1 Processor Type Identification. The *uname* command will now print the machine type and subtype, for example, pdp-11, or vax-780.

3.1.3.2 Interprocess Communication. Interprocess communication routines are now available on the DEC VAX-11/780, and VAX-11/750, and a modified routine package is available for the PDP-11/70. These routines include shared memory, messages, semaphores, and process locking. Also included is an IPC-remove command, *ipcrm* (1) which removes message, semaphore, and shared-memory identifiers from the system.

3.1.3.3 Init and Getty. System V provides the new interface programs *init* and *getty*. These changes affect the operator interface, the system administrator interface and the user interface. (See Section 4.)

3.1.3.4 New General Disk Driver. The new general disk driver will support four different standard disks (RP06, RP07, RM05 and RM80) on a single controller, and it will also support up to four controllers.

3.2 Enhancements and Changes for DEC processors.

3.2.1 KMC/DZ Asynchronous Line Support. The KMC/DZ asynchronous line interface has been changed for the new line disciplines. KMC11-A support has been dropped from this release.

3.2.2 KMC-11 Used More Effectively. Measurements have uncovered a potentially serious performance problem affecting the KMC-11 front-end processor and the DZ-11 line unit. The KMC-11 produced excessive processor load by passing characters to the CPU one at a time. System V software fixes this problem and reduces the host load to reasonable values by ensuring that characters are transferred to the CPU in batches whenever possible. This fix will reduce the processor load on input data from a high of 20 to 40 percent to negligible.

3.2.3 PDP-11/70.

3.2.3.1 Expanded Kernel Space. System V implements a scheme to provide expanded kernel text space by utilizing the PDP-11/70 memory-management hardware. This enhancement will provide space for the introduction of new drivers.

3.2.3.2 Memory Recommendations. The recommended minimum amount of memory for running the UNIX System on a PDP-11/70 is 3/4 megabyte, although the suggested amount is 1 megabyte.

3.2.3.3 Smaller In-Core Inode Table. An inode table cache has been added for the PDP-11/70 so that system data space may be saved.

3.2.4 VAX.

3.2.4.1 Updated Features. System V includes the following updated features for the VAX:

- Power-fail recovery is improved and fully supported.
- New *iltd* and floppy-disk procedures allow booting from RP04/5/6 and RM05 disks.
- A new dump facility writes 5120-byte blocked tapes recorded at 1600 BPI.

3.2.4.2 Memory Recommendations. The recommended minimum amount of memory for running the UNIX System on a VAX is 1 megabyte, although the suggested amount is 2 megabytes.

3.2.4.3 Disk Formatting Utility. System V provides one command, *format* (1M), to format both the RP06 and RM05 disk packs.

3.2.4.4 Vcf Utility. Vcf is a stand-alone utility that tests for the presence of both MASSBUS and UNIBUS[3] devices. The resulting list of addresses is compared with the operating system object module (/unix). Any device address and interrupt vector discrepancies are reported. A complete configuration list can also be generated. See *vcf* (1M) for details.

This utility should be run after any DEC service or any operating-system software change.

3.2.5 Support for New DEC Peripherals.

3.2.5.1 RP07 Disk Drive (VAX Only). The RP07 is a large-capacity, non-removable, high-performance disk drive, built with "Winchester-type" technology. It has a capacity of 705 Mbytes unformatted, 516 Mbytes

3. MASSBUS and UNIBUS are trademarks of Digital Equipment Corporation.

formatted. The average transfer rate is 1.5 Mbytes per second with non-interleaved sectors or 0.75 Mbytes per second with interleaved sectors. The drive is supported as part of the general disk driver (*gd*(7)) on the VAX line only. Two interleaved memory controllers will be required for non-interleaved sector operation.

3.2.5.2 RM80 Disk Drive (VAX Only). The RM80 is a medium-capacity, non-removable, high-performance disk drive. It has a formatted capacity of 124 Mbytes and an average transfer rate of 0.9 Mbytes per second. The drive is supported as part of the general disk driver (*gd*(7)) on the VAX line only.

3.2.5.3 TU78 Tape Drive (VAX-11/780 Only). The TU78 is a 9-track tape drive capable of recording data in phase-encoded format at 1600 bpi or in group-code recording format at 6250 bpi. The tape speed is 125 ips. The drive and controller can be configured with the TU16/TM03 tape drive on the same adapter (see *gt*(7)). The drive is supported by a new combination TU16/TU78 tape driver on the VAX line only. The drive can serve as a boot device.

3.2.5.4 TU77 Tape Drive (VAX Only). The TU77 is a 9-track tape drive capable of recording data at 800 bpi or 1600 bpi at a tape speed of 125 ips. The drive can serve as a boot device for both the 11/780 and 11/750 systems.

3.2.5.5 TS-11 Tape Drive (VAX-11/750 Only). It is a 9-track tape drive that records only in phase-encoded format at 1600 bpi and 45 ips. It is only supported on the VAX-11/750 and may serve as a boot device.

3.2.5.6 ML-11 MOS Memory System. The ML-11 is a fast MOS-memory system that operates in block-mode and emulates an RP06 disk drive. It has a capacity of 1 to 4 Mbytes and a transfer rate of 0.25 to 2.00 Mbytes per second. It is supported as part of the general disk driver (*gd*(7)). The device is not usable as either a boot device or an initial load device. Intended for the PDP-11/70, it should yield faster "swap" and "tmp" operations.

3.3 Languages and Libraries.

3.3.1 C Programming Language. System V includes the following changes to the C programming environment:

- System header files have been reorganized to be mutually exclusive, and some have been enhanced. Therefore, it is advisable to recompile all C programs.
- A new cross-reference program—*cxref*(1)—is designed to replace both *cref*(1) and *xref*(1).
- A new flow-analysis program—*cflow*(1)—will analyze a group of C, YACC, LEX, assembler, and object files and will construct a graph of the external references.
- Change mode *(chmod)* and change owner *(chown)* commands will now fail properly on open files on read-only file systems.

3.3.2 Portable C Compiler. System V changes the machine independent parts of the Portable C Compiler. The major changes deal with obsolete initialization and unsupported operators. For more details, refer to the transition information in Section 4.

3.3.3 C Pre-processor (cpp). The C pre-processor now has a separate manual page (*cpp*(1)) but may only be invoked from the *cc* command. *Cpp* now conforms to the C language specification in the handling of white space and should not be used for general macro processing. (The macro processor *m4* is recommended for general use.)

3.3.4 Cc Command. The *cc* command gives the user greater control over the compilation process with the ability to pass options explicitly to the subprocesses.

3.3.5 Software Generation System (SGS) Performance (VAX only). An SGS is a set of programming tools that support software development for a processor running a UNIX System. It consists of a C language compiler, optimizer, link editor plus a set of utilities that perform object file listing and analysis functions. The compile-time speed of the VAX SGS is improved by making the assembly and link-edit stages more efficient. The link-edit stage is particularly improved by the use of random access archive files (libraries), thus eliminating the lengthy linear searches. The archive redesign also eliminates most of the cases where it was necessary to use *lorder*(1) to order the libraries.

3.3.6 Machine Language Processor (MLP) (VAX only). A new machine language processor using the Common

Object File Format is implemented for the VAX processor. The new MLP has not been implemented on the PDP-11/70 due to address space limitations.

The new VAX object file format is not compatible with the previous object file format (see Transition Problems in Section 4).

3.3.7 Library Performance Enhancements. Many C library routines include performance enhancements, the most notable being improvements in the I/O functions. Some of the changes provided by System V are listed below:

- The buffers of the System V standard I/O library are increased to improve performance.
- The *stdio*(3S) family of string-output functions have been modified to invoke buffering on a per-line basis (to terminals) if full buffering has not been specified. Characters will be collected and then flushed with one *write*(2) system call per string, instead of one call per character, even if non-buffering has been specified.
- Both *putchar*(3S) and *getchar*(3S) now return unsigned character data.
- New 48-bit random number routines for the VAX and the PDP-11/70 have been added. See *drand48*(3C) for details.
- For the *fopen*(3S) routine, the append mode is now enforced strictly. This means that when the 'a' or 'a+' mode is specified, all writes to the FILE will be dynamically forced to the current end-of-file regardless of preceding seeks.
- The *printf*(3S) family of functions has been replaced on the PDP-11/70 with an assembly-language version that is significantly smaller and faster. In addition, *printf* now works correctly in an interrupt routine, even if *printf* was the entity interrupted.
- The *malloc*(3C) routine no longer returns pointers to overlapping regions.
- The *getpwent*(3C) routine now checks for USER IDs greater than MAXUID (as defined in /usr/include/sys/param.h) in /etc/passwd.
- In the external variable *environ*, either the NULL pointer or a pointer to a null string is now an acceptable indication of a null environment.

3.3.8 Profiled Versions of the Libraries. Separate versions of the C library and Math library are available to support user-level profiling. The separate versions allow users to receive profiling information on library functions at the same time they receive profiling information about their own code. These libraries are loaded automatically by *cc -p* on the VAX systems. PDP-11/70 users must specify the files /lib/libp/libc.a and /lib/libp/libm.a explicitly to obtain the profiling information.

3.3.9 Math Library Enhancements. Additions to the math library functions make it easier for users to detect and recover from errors. The library functions distinguish between different types of errors and produce fatal error messages in a few cases. The previous implementations of math library would either core dump or ignore errors. The core dumping problem is replaced with default error messages. In the case of these and other errors the users may choose to supply their own error handling routines and exit or recover with or without error messages.

3.3.10 Expanded Fortran Programming Support Tools. Two new Fortran utility programs have been added: *asa*(1) and *fsplit*(1). *Asa* is used to interpret ASA carriage-control characters, and *fsplit* is used to split F77, RATFOR, or EFL files into separate files.

3.3.11 Fortran 77. Fortran 77 and related commands (*asa, fsplit, efl, ratfor*) are available for the first time on the VAX and PDP-11/70 for System V. Continuing development on the Fortran 77 compiler has improved both the compile-time and the run-time speeds over the version released in System III. However, many of the improvements are only in the VAX implementations. The PDP-11/70 implementation is limited by insufficient address space.

3.3.12 Library Functions. New library functions in the System V release are *dial (3C)*, *erf* (3M), *ftok* (see *stdipc*(3C)), *ftw* (3C), *tempnam* (see *tmpnam* (3S)), *clock* (3C) and five functions to manipulate memory (see *memory* (3C)).

The *assert* (3X) function is changed to invoke *abort* (3C) rather than *exit* (2) when an assertion fails. This provides the user with an automatic "core dump" from which to debug a failure that may be difficult to reproduce.

3.3.13 Other Tools. The *yacc* (1) parser generator has improved optional debugging and produces a faster parser. The symbolic debugger *sdb* (1) is improved. (Sdb (*1*) is not available on the PDP-11/70.) These improvements make the *adb* (1) command on the VAX system obsolescent. The profiling command *prof* (1) has also been improved and the options and output are now the same for all processors.

3.4 Communication and Networking.

3.4.1 Common Synchronous Interface (CSI). CSI provides a standard interface between Level 3 protocols executing as host drivers and Level 2 protocols executing in Programmable Communications Devices (PCDs). CSI makes protocol drivers such as VPM (see below) and the new Level 3 protocol drivers portable across the line of supported processors. The alternative requires writing and supporting individual protocol drivers for each PCD and processor.

3.4.2 Line Disciplines and Virtual Terminal Interface. System V allows substitution of user-written terminal-handling routines for the default system provided routines. This also makes possible the use of a virtual terminal language as one of the sub-line disciplines.

3.4.3 VPM: Virtual Protocol Machine.

3.4.3.1. KMS11 Support. This release of the Virtual Protocol Machine has been enhanced to support not only the KMC11-B microprocessor, but also the KMS11, a DEC microcomputer capable of multiplexing eight synchronous communication links. However, the VPM for the KMS11 does not support BISYNC protocol.

3.4.3.2 Time Stamp on Trace. Time stamping in the trace driver is added to allow better debugging of VPM scripts. (The "stamp of time" on an error during execution is valuable information when trying to position the error within the dynamic program flow.)

3.4.3.3 Power-fail/Restart. The power-fail/restart feature is extended to cover the VPM interface driver. This provides power-fail/restart for all protocols that use VPM (e.g., RJE).

3.4.3.4 The Vpmtest Command. A new command, *vpmtest*, is provided to perform a series of loop-back tests on a user-specified synchronous line. Another new command, *vpmfsr*, executed by *vpmtest* transmits and receives a known data stream and compares the received data stream with the one expected.

3.4.4 Networking.

3.4.5 Uucp. The following enhancements are supplied:

- A silent option for *uux*(1) is included to suppress mail.
- An improved dialing algorithm is included to increase the number of successful connections.
- A job ID is returned by *uucp*. This job ID allows job status and job deletion to be done.
- Forced routing similar to that simulated by *mail*(1) (e.g., uupc x a!b!x) is provided.
- A number of bugs involving execution under the correct user ID have been fixed.

3.5 Commands and Administration Aids.

3.5.1 Tape Back Up. Three new commands are supplied to do faster back-ups and incremental saves.

The command *ff*(1M) performs many of the functions of *find*(1) in generating the path for selected files in a *find* (1) for this application.

The command *finc*(1M) copies selected files to tape in a format much like *volcopy*. Selection criteria (such as "newer than specified file", "untouched in xx days", etc.) reminiscent of *find*(1) can be used. Typically *find* (1M) is faster than the equivalent *find/cpio* commands for large amounts of data. *Finc*(1) tapes are limited to a single tape reel. If an *finc* incremental exceeds one reel, a *volcopy* (1M) backup is recommended.

The command *frec*(1M) performs recovery of files from *volcopy*(1M) or *finc*(1M) tapes. Files to be recovered are requested using their inode numbers. This information is available from the *ff*(1M) output that should be preserved when running any *finc*(1M) or *volocopy*(1M).

Also, the *volcopy*(1M) *command* has a new option, *-buf*, for double buffering I/O. This option provides significant performance improvement.

3.5.2 Screen Editors. The *vi* editor for use with asynchronous CRT terminals is supported.

Vi (1) is a display oriented interactive editor developed at the University of California, Berkeley. It will work on a large number of terminals described in a terminal description file (*termcap* (5)) that is included with the release. The terminal descriptions included in this file are not guaranteed to be correct; some have been used more than others and none has been exhaustively tested. Descriptions of new terminals may easily be added to the *termcap* (5) file simply by editing it. A user adding a new terminal description to the file will also have the responsibility for insuring the accuracy of the entry and testing it out on the terminal. *Vi*(1) has been in use in the field as part of the University of California's 4.1 BSD system (as well as previous releases) and has proven to be an effective tool for both program development and document preparation.

The *termcap* (5) terminal description file is supported.

3.5.3 Accounting. Two new options (i.e., *S* and *E*) are added to *acctcom* to facilitate the new options added to *timex*.

3.5.3.1 Rewritten Timex. The *timex* command is rewritten in a machine-independent way and formatted in the style of the *sar* output. New options for reporting user processes, resource usage and overall system activity are added.

3.5.3.2 Sadp. The *sadp* command reports disk access located and seek distance in either tabular or histogram form.

3.5.4 Line Printer Spooling System. LP is a system of commands that perform diverse spooling functions under the UNIX operating system. Its primary application is off-line printing. LP allows administrators to customize the system to spool to a collection of line printers of any type and to group printers into logical classes in order to maximize the throughput of the devices. Users are provided the following capabilities:

- queuing and canceling print requests,
- allowing or preventing queuing to devices and printing on devices,
- starting and stopping LP from processing requests,
- changing the configuration of printers, and
- finding the status of the LP system.

4. Transition to System V.

System V is an initial load system. The release tapes must be installed as delivered and cannot be used to update an existing system.

4.1 File System Conversion (VAX)

4.1.1 Initial Boot. The System V will be delivered with 1 Kbyte block root(/) and user(/usr) file systems. After the new system is read in and booted, the system administrator may copy local files from the site backups. Any of the standard UNIX system tools may be used to transfer the old files to the new file system. Care should be taken not to overwrite the file system check *fsck*(1) or any of the accounting programs. In general it is wise not to overwrite any of the files delivered with the new system (except those intended for local modification).

4.1.2 Conversion of Existing File Systems. Any of the existing file systems may be converted to the larger block size. Since existing file systems are compatible with the System V, no special conversion tools are necessary. Files resident under an old file system may be copied to a new (1 Kbyte block) file system using *cpio*(1) and *find* (1) or the *finc*(1) and *frec*(1) routines.

4.1.2.1 Cautions. Because of the possibility of additional wasted space on 1 Kbyte block file systems, conversion of full and nearly full 512 byte block file systems may not be possible using the same disk space. A transition tool (file system block analyzer *fsba*(1)) is supplied to summarize the space usage on a small block file system the project the usage on a large block file system.

Also, because *cpio*(1) has a limit to the number of unique linked files it can remember, care must be taken when converting a file system with a large number of these files.

4.1.3 Re-Compilation of Users' Software. Under code and application code that uses the standard I/O library should be recompiled to take advantage of the larger buffer size of the new library. The recompiled code is fully compatible with both file systems and will be more efficient even when using 512 byte file systems.

4.1.4 Modification of Users' Software. User and application code not using standard I/O or doing its own buffering may need to be changed to take advantage of the new file system. Internal buffers used for I/O should be set to be a multiple of 1024 bytes so that they will efficiently match the internal buffer size of the System V. The buffer size may be specified (in a portable way) by using the define constant BUFSIZ from the header file *stdio.h*.

User code and application code that access the file system devices directly must be examined (and probably changed) and recompiled so that they are in accord with the new file system structure.

4.2 *Init & Getty* Interface

4.2.1 Init. The entries in the /etc/inittab (see *inittab*(5)) file have been changed as part of the new interface, so that it is not possible to use the old /etc/inittab file. More states are now possible for spawning programs, and users should read the manual page carefully to select the proper action when a process terminates. Changing from multi-user state to single-user state is done by using *init* as a command, however; the single-user state is now called s (instead of 1) so that the command

init s

brings the system into single user mode. A command file /etc/brc (see *brc*(1M)) is now executed when the system is booted so commands to download devices should be placed here. A similar file /etc/powerfail (see *brc* (1M)) is available for power fail recovery.

4.2.2 Getty. The line speed table is now in the file /etc/gettydefs, so that the sequence of line speeds can be edited on line. (No recompilation of the *getty*(1M) command is necessary.)

The options to getty are slightly different and are documented in the manual pages.

4.2.3 Related Changes. Because of the changes to *init*, the format of the /etc/utmp and /etc/wtmp files are not compatible with previous systems. Users who have written private copies of commands that use these files should be prepared to adopt the new format. Library routines (*getut*(3c)) are provided to ease the conversion.

4.3 Object File Conversion (VAX)

4.3.1 Old Executable Files. Old (pre-System V) executable object files will continue to execute under System V. Only those programs whose functionality is impacted by other changes (see the rest of this section) need to be recompiled.

4.3.2 Old Object Modules. Old (pre-System V) object modules (non-executable object files) are not compatible with the object modules produced by the System V assembler (*as*(1)) or linkage editor (*ld* (1)). They should be replaced by versions recompiled with the System V Software Generation System (SGS).

4.3.3 Old Archive Files. Old (pre-System V) archive files are not compatible with archive files of the System V release. They should be replaced by archives created with the system V *ar* (1) command or converted with the *arcv*(1) command.

4.3.4 Old Libraries. Libraries are archive files; see 4.3.3. above.

4.4 C Source Files

4.4.1 Modification of existing code. The use of old-style initialization (e.g. long × 1) and old-style assignment operators (e.g. =+, =−) will no longer be allowed. All existing code that uses these constructs must be upgraded. The current compilers and *lint* (1) complain explicitly about these constructs.

The use of integer constants with compilers that generate code for 16-bit machines should be reviewed. The compiler now conforms to the language specification for octal and hex constants. The previous implementation erroneously converted octal and hex constants which exceeded the largest signed machine integer to long constants. These constants should only be extended if they exceed the largest unsigned machine constant. Octal and hex constants with the high order (16th) bit on will now have the exact 16 bit pattern and may act like signed negative numbers in expressions.

5. ITEMS NO LONGER SUPPORTED

5.1 DQS on the PDP-11/70

Software using the DQS-11 (BISYNC Synchronous Line Interface) was released for the last time in System III. This means that the UNIVAC and IBM RJE (DQS-based) software will not be included in System V. RJE will be fully supported on the DEC line (KMC-based).

5.2 fget.demon and fget.odemon

The fget.demon.lc and *fget.odemon.lc* commands will no longer be documented in the User's Manual and should not be used except through invocation by *fget*.

5.3 KMC11-A.

A KMC11-A will no longer be supported.

UNIX™ System
Release Description
System V

January 1983

UNIX is a trademark of
Bell Telephone Laboratories, Incorporated.

Copyright© 1983 Western Electric Company, Incorporated.

Portions of this document were copyrighted
1979 Bell Telephone Laboratories, Incorporated.
1980 Western Electric Company, Incorporated.

This document was prepared with specific references to use of the UNIX[1] System on a particular processor, the Western Electric Company 3B20S, which is not presently available except for internal use within the Bell System. However, the information contained herein is generally applicable to use of the UNIX System on various processors which are available in the general trade.

The *vi* (*ex*) editor is based on software developed by the University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

[1] **UNIX** is a trademark of
Bell Telephone Laboratories, Incorporated.

DEC, PDP, VAX, UNIBUS, MASSBUS, DECnet, and SBI
are trademarks of Digital Equipment Corporation.

HYPERchannel is a trademark of Network Systems Corporation.

Ethernet is a trademark of Xerox Corporation.

CONTENTS

1. INTRODUCTION 1
   1.1 Achievements 1
   1.2 Scope 1

2. RELEASE OVERVIEW 1
   2.1 Release Features 2
   2.2 Documentation 3
   2.3 Transition Aids 3

3. PERFORMANCE 3
   3.1 C Language 3
   3.2 Kernel Operations 5
   3.3 Terminal Handling 5
   3.4 Simulated Timesharing Workload 8
   3.5 Administrative Guidelines 9

4. DOCUMENTATION 10
   4.1 Published Documents 10
   4.2 Interim Documents 11

5. ITEMS NO LONGER SUPPORTED 12
   5.1 DQS and Fget 12
   5.2 Floating Point Simulation 12
   5.3 DEC Processors 12

6. TRANSITIONAL INFORMATION 12
   6.1 Release Tapes 12
   6.2 Installation Documentation 12
   6.3 File System Conversion (VAX only) 13
   6.4 Init & Getty Interface 13
   6.5 C Programs 13
   6.6 Vi 13
   6.7 Uucp 13
   6.8 PDP Size Limitations 14
   6.9 Modification Requests 14

7. KNOWN DEFICIENCIES 14
   7.1 Compilers and Language Tools 14
   7.2 Communications 15
   7.3 Lp 16
   7.4 Init and Getty 16
   7.5 File 16
   7.6 Du and Ls 16
   7.7 /etc/checkall 17
   7.8 Mm Macros 17
   7.9 Games 17

8. DISCLAIMERS 18

APPENDICES: 19

Appendix A: New Pages for the *UNIX System User's Manual - System V* and the *UNIX System Administrator's Manual - System V*

Appendix B: Setting Up the UNIX System (DEC)

Appendix C: System V Init and Getty

Appendix D: An Introduction to Display Editing with Vi

Appendix E: Ex Reference Manual

Appendix F: Uucp Tutorial

Appendix G: Uucp Administrators Manual

Appendix H: Installing an Auto Call Facility on a UNIX System

Appendix I: UNIX System Modification Requests Resolved in UNIX System V

LIST OF FIGURES

Figure 1. Terminal Output - Raw Mode 6
Figure 2. Terminal Output - Cooked Mode 7
Figure 3. Terminal Data Loop-Around - Raw Mode 8
Figure 4. Terminal Data Loop-Around - Cooked Mode 9
Figure 5. UNIX System V Throughput vs. Applied Load 10

LIST OF TABLES

TABLE 1. Normalized Object Code Execution Speed 4
TABLE 2. Normalized Object Code Text Size 4
TABLE 3. Normalized Compilation Speed 4
TABLE 4. Kernel Function Execution Time in msec (III time / V time) 5
TABLE 5. Peak Output Rate in kilobaud 6
TABLE 6. Maximum Loop-Around Rate in kilobaud 7
TABLE 7. Peak Throughput in 1000 jobs/hour 8
TABLE 8. State of /usr/games 17

1. INTRODUCTION

UNIX[1] *System V* is being provided for Digital Equipment Corporation (DEC[2]) VAX[2] and PDP[2] processors.

UNIX System V is provided for the following processors:

- DEC VAX-11/780 and VAX-11/750
- DEC PDP-11/70

and includes features in the following general categories:

- Kernel Enhancements and Changes
- Enhancements for DEC Processors
- Languages and Libraries
- Communication and Networking
- Commands and Administration Aids

1.1 Achievements

The single most noticeable achievement in this release is the significant gain in overall system performance. This improvement translates directly into improved system response time for the user, or an increased capacity in users per system.

System V provides UNIX System users with timely and useful documentation. The standard documentation is included as part of the release and is automatically distributed with the software.

1.2 Scope

This *System Release Description* (*SRD*) contains a current list of the *System V* features, and current information on documentation and transitional (*System III* to *System V*) concerns. It is intended to provide an overall description of *System V*, while emphasizing the system performance improvements, new documentation, and transitional information.

Section 2 provides a general overview of what is in this release and what is provided with it. A detailed performance evaluation of *System V* is given in *Section 3*. *Section 4* describes the documentation available with *System V*.

Sections 5, 6, and *7* document items no longer supported, transitional aids, and known deficiencies, respectively. This information is intended to ease the installation of this release and the transition of software and users to *System V*. Please read these sections before attempting to install this release. Also, please read the documents referenced in these sections, as they will contain many helpful hints that will make the installation job easier.

2. RELEASE OVERVIEW

This section provides an overview of *System V*.

---

[1] UNIX is a trademark of Bell Telephone Laboratories, Incorporated.

[2] DEC, VAX, and PDP are trademarks of Digital Equipment Corporation.

2.1 Release Features

The following is a summary list of the individual features in *System V*.

2.1.1 Kernel Enhancements and Changes

2.1.1.1 Operating System Performance. The performance of the operating system on the VAX systems has been improved by a larger block file system, eliminating redundant work, and providing a larger number of I/O buffers. The new features are:

- 1 Kilobyte Block File System
- Physical I/O Buffer Facility
- Larger Buffer Pool
- File Table Free List
- Faster *fork* System Call
- Faster *open* System Call (inode hashing)
- Hashing of the *sleep/wakeup* Mechanism (also on PDP)

2.1.1.2 File System Changes. The file system has been enhanced to maintain the consistency of the disk information. This eliminates a class of disk errors associated with system crashes. The new features are:

- Ordered Writes (VAX only)
- Dual *fsck* Capability
- Faster *fsck* Option (VAX only)
- Improved File System Access
- *fuser* Command

2.1.1.3 Operating System Enhancements. System V provides new features including:

- Processor Type Identification
- Interprocess Communication
- *init* and *getty*
- New General Disk Driver

2.1.2 New Hardware Enhancements

The UNIX System running on the Digital Equipment Corporation processors has been updated to support new devices introduced by the hardware vendor. The new devices are:

- RP07 - Large Capacity (516 Megabyte) Disk Drive (VAX Only)
- TU78 - High Speed (6250 BPI) Tape Drive (VAX-11/780 Only)
- RM80 - Medium Capacity (124 Megabyte) Disk Drive (VAX Only)
- TS11 - Medium Speed (1600 BPI) Tape Drive (VAX-11/750 Only)
- ML11 - MOS Memory

2.1.3 Languages and Libraries

The language tools of *System V* have been changed to provide run-time efficiency, and improved performance. In particular, the C library has been changed to take advantage of the improvements in the operating system and to perform its functions more efficiently. The list of features includes:

- System Generation Software (SGS) Performance Improvements (VAX only)
- Improved Machine Language Processors (MLP) (VAX only)
- Library Performance Enhancements
- Larger Buffer Size
- Profiled Versions of the Libraries
- Math Library Enhancements
- New Library Functions

2.1.4 Communication and Networking

The Communications and Networking Facilities of *System V* provide enhanced support of networking across all supported processors. Kernel and networking program enhancements are

provided for:

- *uucp*
- *Common Synchronous Interface* (CSI)
- *Remote Job Entry* (RJE)

2.1.5 Commands and Administration Aids

System V provides the user and administrator with uniform capabilities across the supported processors. This list includes:

- Incremental Tape Back-up
- *vi*[1]/*ex*[1] Screen Editor Support
- Accounting Enhancements
- System Activity Report
- Line Printer Spooling System

2.2 Documentation

The new documentation highlights are:

- System Error Message Manual
- F77 library documentation

The release documentation is described in *Section 4* of this *SRD*.

2.3 Transition Aids

Customers are required to perform some conversion functions when installing *System V*. A guide to aid in the transition from *System III* to *System V*, *UNIX System Transition Aid - System V*, contains documents concerned with:

- Converting to a 1 Kilobyte Block File System
- UNIX System Archive Format Changes
- User Level Changes to the C Language
- The Common Object File Format
- UNIX Operating System Environment Changes

See *Sections 5, 6, and 7* for more details.

Conversion tools will be provided. These include:

- *arcv*(1) - convert archive files from PDP-11 to common archive format
- *convert*(1) - convert object and archive files to common formats
- *fsba*(1) - file system block analyzer
- *non-btl*(1) - re-install mm macros without Bell Labs specific features

3. PERFORMANCE

System V offers significantly improved performance on the 32-bit VAX family machines relative to System III. Although the magnitude of the improvement is dependent on workload, VAX time-sharing installations should experience an approximate increase in capacity of 26%. These gains are the result of changes in the UNIX System kernel, the C compiler and libraries.

3.1 C Language

The following results are based on a collection of small (about one hundred lines average) C language programs which do not use library functions. The faster compilation times reflect changes to the kernel and improvements in many C library functions.

---

[1] The *vi*(*ex*) editor is based on software developed by the University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

3.1.1 Object Code Execution Speed

VAX-11/780 and PDP-11/70 object code speeds are essentially unchanged from System III. In Table 1, object code execution speed factors are normalized to the VAX-11/780 by dividing the 11/780 times by the respective times for each table entry. Larger numbers indicate faster program execution.

Machine	*Speed*
VAX-11/780	1.00
VAX-11/750	0.64
PDP-11/70	0.85

TABLE 1. Normalized Object Code Execution Speed

3.1.2 Object Code Size

The following relative size results were obtained by subtracting the size of a null program object from the sizes of the benchmark object modules and dividing by the VAX-11/780 result. In the table, smaller numbers indicate more compact storage on the respective machine. System V object code text size on the VAX-11/780 is about the same as that of System III. PDP-11/70 System V text size is slightly smaller than for System III.

Machine	*Size*
VAX-11/780	1.00
VAX-11/750	1.00
PDP-11/70	1.04

TABLE 2. Normalized Object Code Text Size

3.1.3 Compilation Speed

The following table shows compilation speed factors normalized to the VAX-11/780 System V result. Larger numbers indicate faster program execution. Compilations of small programs on the VAX machines take place about twenty percent faster using System V. Table 3 figures are rough guides only, since program size and use of library functions affect compilation times differently on the different machines.

Machine	*Speed*
VAX-11/780	1.00
VAX-11/750	0.65
PDP-11/70	0.73

TABLE 3. Normalized Compilation Speed

3.1.4 Libraries

Significant performance improvements were made in the following C library I/O functions: *getc*(3S), *putc*(3S), *gets*(3S), *puts*(3S), *fgets*(3S), *fputs*(3S), *fread*(3S) and *fwrite*(3S). For small numbers of bytes handled, the improvements tend to be minor; for larger operand strings, however, the revised functions are up to a factor of five faster.

3.2 Kernel Operations

The operating system kernel for the VAX-11/780 is much faster and is responsible for much of the improvement observed for most commands. Table 4* lists the times in milliseconds for a variety of kernel functions along with the improvements (in parenthesis) calculated as System III time divided by System V time.

Operation	*VAX-11/780*	*VAX-11/750*	*PDP-11/70*
syscall (*getpid*)	.18 (1.33)	.30	.23 (0.96)
context switch	.71 (1.37)	1.02	1.03 (1.16)
chdir '.'	1.22 (2.49)	1.92	3.35 (0.95)
open/close 'file'	2.47 (2.77)	3.85	6.82 (0.95)
search path 3 lev.	5.93 (2.94)	8.96	17.1 (0.95)
search dir. 32 pos.	3.03 (2.48)	5.11	11.1 (0.97)
pipe 512 bytes	3.21 (1.14)	5.05	5.69 (0.99)
pipe 1024 bytes	3.91 (1.30)	5.66	8.61 (1.00)
read 1 - buf cache	.94 (1.11)	1.56	1.42 (0.97)
read 512 - buf cache	1.38 (1.10)	2.05	2.51 (0.97)
read 1024 - buf cache	1.81 (1.32)	2.57	4.17 (0.99)
access disk block	3.10 (1.00)	4.69	4.19 (0.88)
read 4K file	19.4 (4.1)	34.7	66.3 (0.88)
fork/exit 8K data	21.6 (1.07)	28.3	23.9 (0.98)
exec 8K bss	18.5 (1.23)	25.7	34.6 (1.08)
exec 8K data	25.5 (1.41)	37.2	51.7 (1.03)
exec 512 arglist	61.7 (1.02)	94.8	100 (1.02)

TABLE 4. Kernel Function Execution Time in msec (III time / V time)

In Table 4, context switches are faster largely as a result of an improved *sleep/wakeup* mechanism. The faster VAX file operations are attributable to improved file and inode table searches.

The improvements in read and pipe operations for 1024 byte units on the VAX are mainly the result of the switch to the larger 1024 byte block size. Note that a single 1024 byte transfer takes place much faster than a pair of 512 byte operations; user application programs that currently do I/O in units of 512 bytes will obtain much better performance if changed to perform 1024 byte transfers. For programs that do I/O via the C library, this is taken care of automatically when the programs are recompiled. Although the time for disk block access has changed relatively little, disk accesses now transfer twice as much data; the improved performance is demonstrated by the shorter CPU times to read a 4096 byte file. The *exec* performance improvements are also due primarily to the larger block size.

3.3 Terminal Handling

Figures 1 through 4 and Tables 5 and 6 show the output and loop-around bandwidths for System V expressed in kilobytes per CPU second. CPU utilization for terminal handling on a production system can be derived as the ratio of observed bytes per second on the system in question to the corresponding bandwidths from Figures 1 through 4.

---

* The results for *chdir*(2), *open*(2)/*close*(2), path search and directory search are for 120 inode table slots in use on the VAX-11/780 and 80 slots in use on the VAX-11/750 and PDP-11/70.

3.3.1 Terminal Output

The results in Figures 1 and 2 were obtained by writing data at 9600 baud from the system to twenty terminal lines. The VAX-11/780 with KMC/DZ assist yields very high bandwidths. For the VAX-11/780 without KMC/DZ and the PDP-11/70, the required CPU involvement in character processing yields lower bandwidths overall, especially in "cooked" mode. The poor showing of the VAX-11/780 without KMC/DZ compared to the PDP-11/70 results from the interrupt-per-character nature of the DZ-11 as opposed to the DMA property of the DH-11 used with the PDP-11/70.

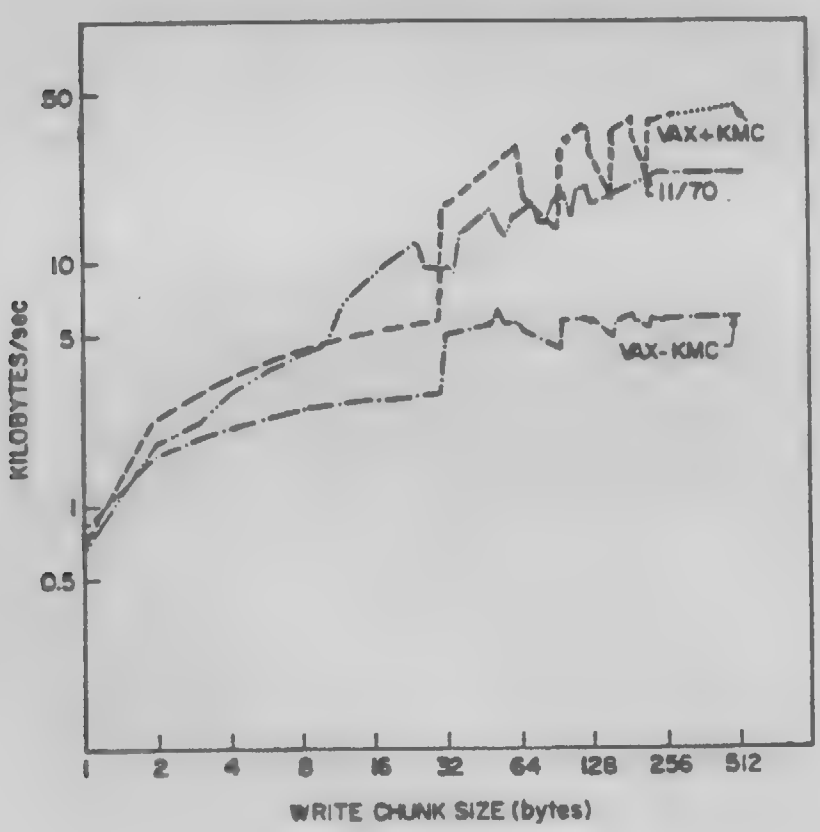


Figure 1. Terminal Output - Raw Mode

The peak output rates for System V are summarized in Table 5

Machine	*Cooked*	*Raw*
VAX-11/780+KMC	40.1	40.3
VAX-11/780-KMC	2.9	5.6
PDP-11/70	6.1	23.7

TABLE 5. Peak Output Rate in kilobaud

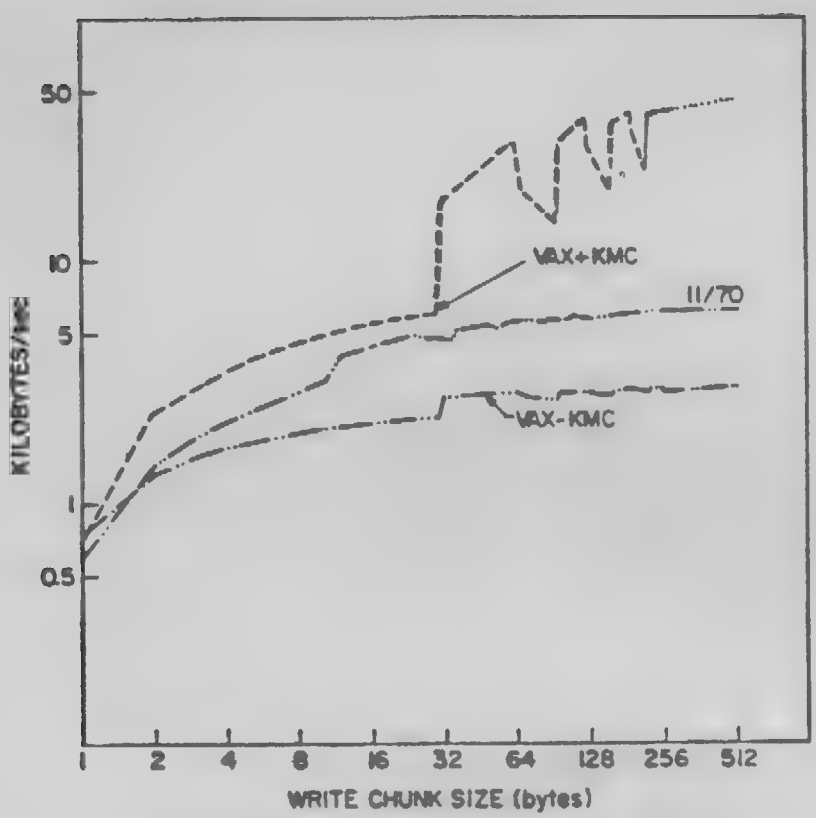


Figure 2. Terminal Output - Cooked Mode

3.3.2 Terminal Data Loop-Around

The loop-around bandwidths in Figures 3 and 4 were obtained by attaching special hardware to ten lines which circulated the outgoing data back to the input. The bandwidths in these figures represent input limitations, since the output-only bandwidths of Figures 1 and 2 are much greater.

Table 6 summarizes the maximum bandwidths obtainable for data loop-around. Bandwidths for the VAX-11/780 with KMC/DZ system are up to an order of magnitude better than for System III as the result of revision of the KMC driver, which now avoids unnecessary character-at-a-time interrupts by the KMC.

Machine	*Cooked*	*Raw*
VAX-11/780+KMC	2.4	4.2
VAX-11/780-KMC	1.1	1.7
PDP-11/70	1.5	2.3

TABLE 6. Maximum Loop-Around Rate in kilobaud

Results indicate that the VAX-11/780 with KMC and PDP-11/70 systems are unlikely to be burdened by CPU overhead from terminal handling. Therefore, no restrictions need be placed on the use of 9600 baud lines for normal interactive use.

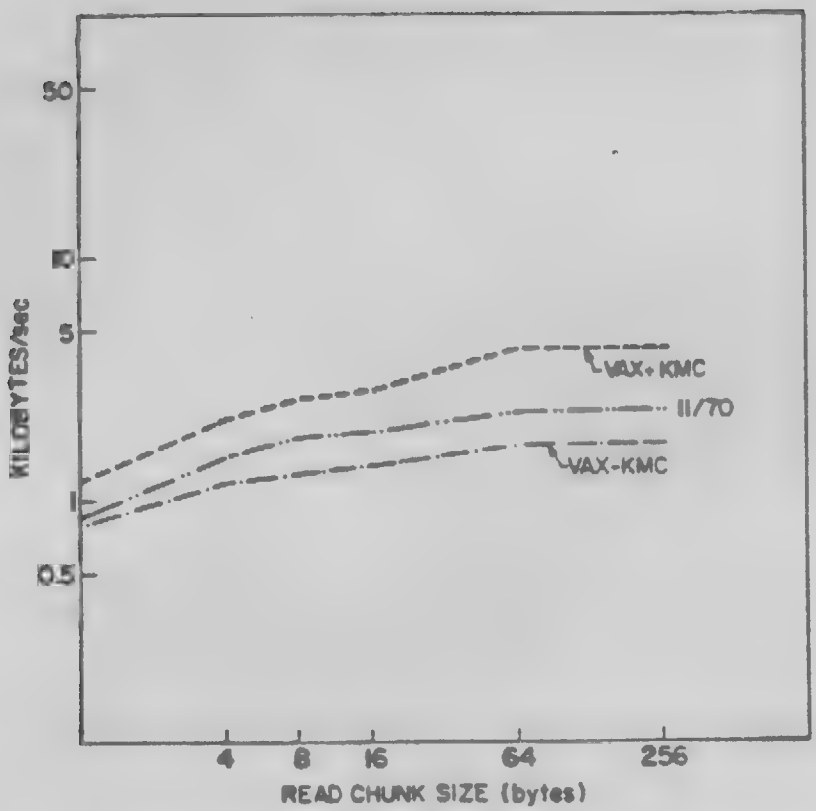


Figure 3. Terminal Data Loop-Around - Raw Mode

3.4 Simulated Timesharing Workload

The time-sharing simulation benchmark used measures the throughput of the system by executing shell scripts comprised of UNIX System commands. These scripts are modeled on measurements of a program development environment. An increasing number of scripts are executed concurrently to test the system under a variety of load conditions. Each script executes about forty of the commands observed to consume the most CPU time in the time sharing environment.

The results given in Figure 5 and Table 7 are expressed as throughput in jobs per hour versus load. With light loads, the system is partially idle waiting for disk I/O. With the heavier loads, system overhead increases due to reduced buffer cache hit ratio and swapping (although the VAX machine had sufficient memory so that swapping did not occur). Estimated peak throughputs of the various systems are compared in Table 7.

Machine	*III*	*V*	*(% improvement)*
VAX-11/780	7.74	9.75	(+26%)
VAX-11/750	n/a	6.01	(n/a)
PDP-11/70	5.64	5.83	(-3%)

TABLE 7. Peak Throughput in 1000 jobs/hour

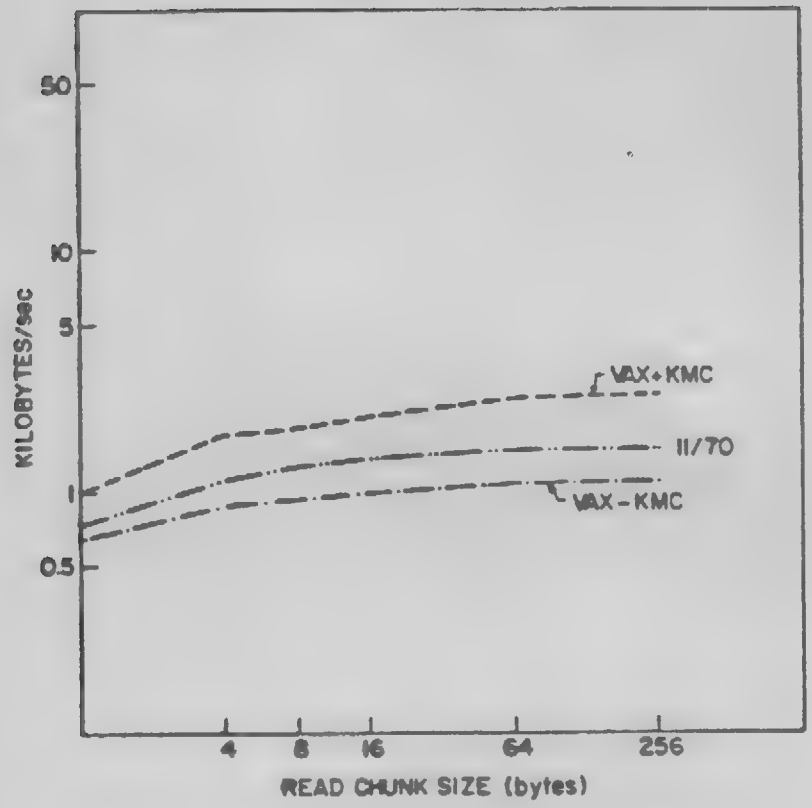


Figure 4. Terminal Data Loop-Around - Cooked Mode

The VAX-11/780 yielded a gain of 26%, while the PDP-11/70 showed a slight decrease of 3%. Table 7 throughput gains translate into corresponding increases in the number of users that can be supported in a program development environment.

3.5 Administrative Guidelines

The number of timesharing users that can be supported depends on a number of factors including the type of work, fraction of users active as opposed to simply logged-on, quality of service desired and willingness to limit heavy resource consuming activities. The following guidelines for typical program development environments are based on interviews with computer center customers with expertise in machine sizing; they are, necessarily, rough estimates.

Experience indicates that approximately thirty-five users logged-in and active gives a reasonable response time compromise for VAX-11/780 System V systems with 3 disk drives. VAX-11/750 and PDP-11/70 machines will support about two-thirds the load of the VAX-11/780, assuming two drives for the VAX-11/750 and three for the PDP-11/70. Use of higher loading is feasible with activity restrictions and willingness to tolerate poor response time. On the other hand, lighter loadings will yield improved response. The ordinate in Figure 5 is a measure of system capacity; experience shows that 10,000 jobs/hour in this figure corresponds to roughly thirty-five (reasonably) satisfied users.

Throughput in Table 7 for System V on the VAX-11/780 is for a system configured with two hundred system buffers. Installations with two or more megabytes of memory may get a performance increase of a few percent relative to Table 7 by configuring with a larger number of buffers.

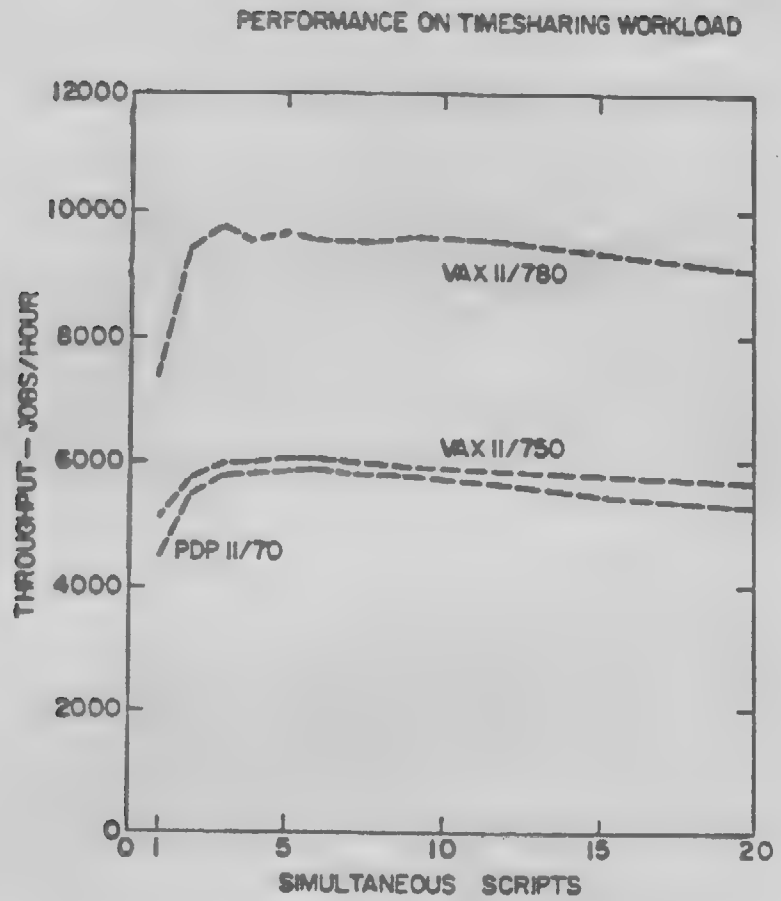


Figure 5. UNIX System V Throughput vs. Applied Load

System buffers and user programs both contend for main memory, so that the optimum number of buffers is a compromise. The number of buffers may be increased until swapping overhead becomes significant. The administrator can obtain the optimal balance for a particular system by monitoring system activity using *sar*(1) and reducing the number of configured buffers when a significant increase in swapping is observed.

The results presented reflect the improved performance obtained by setting the "sticky" bits on the benchmark commands. System administrators should set the "sticky" bits on the commands that are heavily used at their particular installation. Results with "sticky" bits off were five to ten percent poorer.

4. DOCUMENTATION

4.1 Published Documents

There are two major parts to the standard documentation package: a the guides, and the manuals. The following is a list of the guides and manuals applicable to *System V*:

System V Documentation

UNIX System Release Description—System V
UNIX System Transition Aid—System V
UNIX System User's Manual—System V
UNIX System Administrator's Manual—System V
UNIX Operating System Error Message Manual—System V
UNIX System Administrator's Guide
UNIX System Operator's Guide
UNIX System User's Guide
UNIX System Document Processing Guide
UNIX System Graphics Guide
UNIX System Programming Guide
UNIX System Support Tools Guide

One copy of these documents will be provided to the purchaser of a UNIX System license. Additional copies may be ordered through the sales contact at AT&T Technology and Licensing.

4.2 Interim Documents

Additional documentation not yet incorporated in the other *System V* manuals or guides is being distributed in the appendices of this *SRD*.

4.2.1 New Manual Pages

Appendix A provides new manual pages for the *UNIX System User's Manual* and the *UNIX System Administrator's Manual*. These new manual pages are not found in the published UNIX System manuals.

The manual pages of special interest are highlighted here. See the manual pages *ex*(1), *vi*(1) and *termcap*(5) for the new text editor *vi*. *Fsba*(1M) is a new manual page for a tool used in converting to a 1K block file system.

4.2.2 Setting Up the UNIX System

The *UNIX System Administrator's Guide* contains a section for loading the system tapes and setting up a new UNIX system. The new *init* and *getty* programs have invalidated the installation procedures in this section. **A revised section which must be used for setting up the DEC systems during the installation of *System V* is provided in *Appendix B*.**

4.2.3 UNIX System V Init and Getty

The memorandum *UNIX System V Init and Getty* is provided in *Appendix C*. This memorandum describes the interface and features of the new *init* and *getty* programs. It provides useful transitional information along with the corresponding manual pages in the *UNIX System Administrator's Manual*.

4.2.4 Screen Editors

Two documents have been included for the new *vi*(1) and *ex*(1) commands. *An Introduction to Display Editing with Vi* (*Appendix D*) is a tutorial introduction to the screen editor. It also contains a summary table of all the commands for reference. The *Ex Reference Manual* (*Appendix E*) describes the line oriented text editor.

4.2.5 Uucp Documentation

Appendix F contains the *Uucp Tutorial* and *Appendix G* contains the *Uucp Administrator's Manual*. These two documents describe how to set up and use the *uucp*(1C) system.

4.2.6 ACU Installation

Appendix H contains the memorandum *Installing an Auto Call Facility on a UNIX System*. This information should be especially useful for setting up Auto Call Units (ACU) to be used with *uucp*(1C) and *cu*(1C).

4.2.7 Manual Pages

The manual pages for the published version of the *UNIX System User's Manual* and the *UNIX System Administrator's Manual* are contained on the system in the directories **/usr/man/u_man** and **/usr/man/a_man**. The manual pages provided in *Appendix A* are to be found in the directory **/usr/man/local**. This arrangement allows the *man*(1) command to access the latest manual page while keeping the published manual pages separated from the updated pages. The system administrator may wish to move these updated manual pages from **/usr/man/local** to their usual home in either **/usr/man/u_man** or **/usr/man/a_man** at installation time so that the site local manual pages can be placed in **/usr/man/local**.

The pages for the new *UNIX Operating System Error Message Manual* are contained in the directory **/usr/docs/errmanual**. This manual is not obtainable by use of *man*(1). However, *troff*(1), *nroff*(1), and *man*(5) can be used to obtain the formatted manual pages.

5. ITEMS NO LONGER SUPPORTED

5.1 DQS and Fget

The list of items no longer supported includes:

- DQS (on the PDP-11/70)
- *fget.demon* and *fget.odemon*

5.2 Floating Point Simulation

Floating point simulation is no longer supported for any processor. This means *System V* will only run on PDP processors with floating point hardware.

5.3 DEC Processors

DEC processors which are no longer supported include:

- PDP-11/23
- PDP-11/34
- PDP-11/45

6. TRANSITIONAL INFORMATION

The information provided here is intended to ease the transition from *System III* to *System V*. Additional information is provided in the document *UNIX System Transition Aid - System V*, which should also be consulted.

The transitional information included here is applicable to all supported processors unless otherwise noted.

6.1 Release Tapes

This release is being provided as a set of initial load tapes for each of the supported VAX and PDP processor lines. The following tapes constitute the initial load tapes provided for *System V*:

Tapes for the VAX-11/780, VAX-11/750, and PDP-11/70:

- Boot Loader + *root* File System + Selectables (Tape 1)
- User File System (Tape 2)

The detailed format and contents of these release tapes can be found in the document *Setting up the UNIX System (DEC)*. See *Appendix B*.

6.2 Installation Documentation

Installation procedures that have been revised since the their last formal publication are included in *Appendix B*. This revised document must be used to install this release. **Do not use the "setting up" procedures in the published** *UNIX System Administrator's Guide* **to install the system.**

6.3 File System Conversion (VAX only)

The procedures for converting to a 1K block file system are provided in the document *UNIX System Transition Aid - System V*. A tool to aid in the conversion, the file system block analyzer *fsba*(1M) is provided in *System V*.

6.4 Init & Getty Interface

The new *init* and *getty* are described in *Appendix C*. This document and the associated manual pages (*init*(1M), *getty*(1M), *gettydefs*(4), *inittab*(4), *brc*(1M), and *shutdown*(1M)) should be read before attempting to install *System V*.

When making changes to the *shutdown* procedure (e.g., while "setting up" the system), make sure the *telinit* command in the *shutdown* procedure is the last command in that procedure; i.e., no other commands can follow *telinit* for this procedure to work correctly.

6.5 C Programs

The handling of enumeration data types has been corrected. Previously an *enum* variable would be stored in the smallest data type that would hold it; now it is always stored in an integer. This makes newly compiled object files incompatible with those compiled with a UNIX System before *System V*. Thus, all local library functions (and other object modules) employing enumeration data types must be recompiled to prevent variable size incompatibilities when they are link-edited (*ld*(1)).

The following change has been made to the *scanf*(3S) functions. If a "%[" specification fails to match an input character, the function returns with a mismatch. In the previous version, this was treated as a match and a string consisting only of the *NUL* character was returned. The new behavior conforms to that of other UNIX System implementations.

Programs that include *utmp.h* must be recompiled. Some structure definitions in *utmp.h* have been changed. Some code changes may also be required.

On the VAX, if one of the source files for a program is changed, a re-compilation of all source files will be required before attempting to link edit, *ld*(1), the object files. This is necessary because of the implementation of the new *Common Object File Format*.

6.6 Vi

If *make*(1) is used to rebuild the *vi*(1) command and it terminates prematurely for any reason, the following precaution must be taken. The "make" of *vi*(1) must be restarted only after running the command

make clean

This is because all error messages and strings are extracted from the source code and placed in a common library of messages and strings. This library resides in **/usr/lib/ex3.7strings**, once the *make* has completed. There is a good chance this library will be corrupted if the *make* terminates prematurely.

6.7 Uucp

A new feature of the *uucp*(1C) command is the ability to specify a routing sequence within a source or destination file argument. This forwarding feature allows files to be passed through several machines before arriving at the destination machine. Users must be aware that this feature will be blocked by any machine in the routing list that is not set up to allow forwarding and that is not running *System V*.

When setting up *uucp*(1C) while installing *System V*, the files **/usr/lib/uucp/L_stat** and **/usr/lib/uucp/R_stat** must be removed. These files have changed from ASCII to binary. If they are not removed, the *uustat*(1C) command will fail.

The names of the *uucp*(1C) *work* files were increased by one character to provide job tracking via *uustat*(1). *Uucp*(1C) will not recognize old work files. Therefore, these files **(C.*)** must be removed from the spool directory **/usr/spool/uucp**.

The **-m** option will not do anything when *uucp*(1C) is executed via the **-e** option (remote execution option). *Uulog*(1C) no longer produces any output unless an option is specified on the command line. *Uuclean*(1M) should be used on valid *uucp*(1C) spool directories only.

A tutorial on *uucp*(1C) and a manual on the usage and administration of a *uucp* network are included as appendices to this document, and should be consulted for other details. See *Appendix F* and *Appendix G*.

6.8 PDP Size Limitations

The address space in the operating system on the PDP processors is limited. Loading drivers, such as VPM, may require changes to the operating system makefiles. See the *Setting Up the UNIX System (DEC)* document in *Appendix B* for details.

6.9 Modification Requests

A summary of UNIX System modification requests (MRs) resolved in *System V* is provided in *Appendix I*.

7. KNOWN DEFICIENCIES

The following deficiencies are known to exist in *System V* and are itemized here to allow the user to avoid them until they are fixed in a future UNIX System release.

7.1 Compilers and Language Tools

7.1.1 C Compilers

The compiler currently does not detect a shortage of temporary file space in the **/tmp** or **/usr/tmp** directory and will report compilation errors when this condition occurs. This situation will be diagnosed properly in the next release. The user can specify a directory for temporary file space by setting and exporting the environment variable *TMPDIR*.

On the VAX systems, if an unsigned integer or unsigned long integer with the high order bit set is converted to **double** or **float**, a negative value is produced instead of a large positive value. This will be fixed in the next release.

On the VAX systems, casting a *float* or *double* to *long*, *int*, *short*, etc., followed by a mask, will not be optimized correctly if the mask consists of contiguous low order bits. For example:

```
double d = 1.7;
long l = ((long)d) & 0x7FFFFFL;
```

On the VAX systems, when using bit fields, if one of the fields happens to be the high-order 16 bits of a word, accesses of the field will be optimized incorrectly. For example:

```
struct
{
unsigned low:16;
unsigned high:16;   /* field will be accessed wrong */
} x;
```

For compilers or cross-compilers running on the PDP-11/70 processor, structures and **automatic** arrays are limited in size to fewer than 4096 bytes.

7.1.2 F77 Compiler

On the VAX systems, a relational expression with the integer constant "0" on the left hand side (e.g., "if (0 .gt. i) ...") is optimized incorrectly. To eliminate the problem, reverse the expression so the "0" is on the right-hand side. This will be fixed in the next release.

On the VAX systems, the **-o** option doesn't work as documented. It must appear after the last filename on the command line, not before.

7.1.3 Machine Language Processors

On the VAX systems, the **-e** option for the link editor *ld*(1) doesn't work. These problems will be dealt with in the next release.

On the VAX systems, if *strip*(1) is to be run against an archive that has been built from executable files (as opposed to unlinked ".o" files), the *-r* option must be used. Failure to do so will damage the archive without a warning. Also, the little used options **-l**, **-s** and **-x** do not work properly and should not be used. These problems will be corrected in the next release.

7.1.4 Libraries

The *abort*(3C) function doesn't close (flush) files, so some output may be left in internal buffers. This will be changed in the next release.

The values returned by *ftell*(3S) on a file opened for appending (i.e., with type **a** or **a+**) may be confusing. For further information, see the manual page for *fopen*(3S).

The value returned by *log*(3M) and *log10*(3M) on non-positive input is -HUGE, not 0 as documented.

If a program opens more than one stream simultaneously with *popen*(3S), it must close them in the reverse order (most recently opened first).

The commands "cc -p ..." on the PDP and "f77 -p ..." on all supported processors do not automatically load the profiled versions of the libraries. They must be named explicitly on the command line: **/lib/libp/libm.a** (Math Library) and **/lib/libp/libc.a** (C Library).

7.1.5 Debugging Tools

The *lint*(1) analyzer has trouble with *enum*s as function arguments.

The profiler *prof*(1) fails if the program text is larger than 64K bytes. This limitation will be removed in the next release.

The cross-referencer *cxref*(1) considers each appearance of a formal parameter in a macro definition as a definition of that parameter which may therefore appear many times. Also, the temporary files created may exceed the system limit, leading to incomplete cross-reference listings. The program will diagnose this failure clearly in the next version.

7.2 Communications

7.2.1 Uucp

The **-r** option is missing from the *uucp*(1C) manual page. This option delays the execution of the *uucico* daemon process, and is useful in *shell* scripts that invoke many *uucp*(1C) commands. The option should be specified on all but the last *uucp*(1C) command so that only one invocation of *uucico* occurs.

7.2.2 Uuname

If the entry for a system in **/usr/lib/uucp/L.sys** exceeds 128 characters, the output of *uuname*(1) will contain some extraneous information which can be used to compromise system security. A solution to the problem is to re-make **/bin/uuname**, after editing **/usr/lib/cmds/uucp/uuname.c** to increase the size of the array **s** to 300 (from 128), and increase read byte limit of the *fgets*(3) subroutine calls to 300 (from 128). The *make* file for *uuname*(1) is **/usr/lib/cmds/uucp/uucp.mk**.

7.2.3 Cu

The *cu*(1C) manual page will be altered in the next release to clarify the use of the *-s* and *-l* options. When using a dial line, the speeds available using a 212A data-set are 300 and 1200 baud; the default speed is 300. If a speed is specified using the **-s** option and if it is less than or equal to 300, 300 will be used; if the speed is greater than 300, then 1200 will be used. When a direct line is used, the speed specified by the **-s** option is used. If the **-l** option is used, the **-s** option is ignored and the speed is found in the *L-devices* file for the corresponding line.

The manual page will be changed in the next release to restate the meaning of the input sequence *~%nostop*. Currently the manual page states that *~%nostop* command turns off the input control for the remainder of the terminal session. This is incorrect. The *~%nostop* command toggles between DC1/DC3 input control and no input control.

7.2.4 VPM

If the *vpmstart*(1M) command is included in **/etc/brc**, then the error message "getopt: not found" will be printed on the system console when the system is booted. This is the result when *vpmstart* attempts to invoke *getopt*(1) before the **/usr** file system is mounted (*getopt*(1) is normally located in **/usr/bin**). To avoid this problem, move (*mv*(1)) *getopt* from **/usr/bin** to **/bin**.

7.3 Lp

If the user attempts to print a non-ASCII file using *lp*(1), the result will be an error message of the form "xxxx Bus Error - core dumped." In the next release, the *lp*(1) program will be modified to filter (remove) non-ASCII characters.

If an error message about the file *outputq* results when attempting to use *lp*(1), perform the following file creation command:

```
cp /dev/null /usr/spool/lp/outputq
```

A problem exists that may occasionally result in the wrong file being printed. If a user tries to print a file that happens to have the same name as a file in the spool directory (**/usr/spool/lp**), then *lp*(1) will print the file from its own directory instead of the users file. Temporarily rename your file to print it.

The *hp2631a* filter distributed with *lp*(1) randomly drops characters from the printout. This can be cured by changing *termio*(4) options in the filter program by adding the following line:

```
tbuf.c_cc[VMIN]=1;
```

If the error message "Model non-existent" occurs when a user tries to copy a model interface program using the *lpadmin*(1M) command, the files in the **/usr/spool/lp/model** directory should have the following permissions, owner, and group:

```
Permissions: -rw-r--r--
Owner: lp
Group: bin
```

The *accept*(1M) and *reject*(1M) commands always return a successful indication even when they fail.

7.4 Init and Getty

Init(1M) uses the pathname **/dev/syscon** (which is normally linked to **/dev/console**) for output to the console. If "init s" is executed from a terminal (e.g., in *shutdown*(1M)) other than **/dev/console**, **/dev/syscon** will be linked to *that* terminal. Typing the *delete* key from **/dev/console** returns control to **/dev/console** only if the *remote* console has hung up.

7.5 File

While using *file*(1) to determine the type of an object file on the PDP-11/70, it will be inaccurately reported that the file is "4.0 pure executable". It should be reported that the file is "5.0 pure executable". The problem is associated with the *file*(1) command, not the object file.

7.6 Du and Ls

The size, in blocks, of a file or file system can be incorrectly reported by *du*(1) and *ls*(1) *(using the -s option)*. Sizes returned by these commands are multiples of 512-byte blocks. Therefore, for 1024-byte block files or file systems, sizes should always be even numbers. However, a one block error in the size can occur because of a flaw in the algorithm used to determine the size.

7.7 /etc/checkall

The prototype **/etc/checkall** procedure included on the *System V* installation tape is incorrect. The *-T* option to be passed to *dfsck*(1M) should be *-t* (lower case).

7.8 Mm Macros

The *System V mm*(1) macros have Bell Laboratories specific features which should be removed. The *non-btl*(1) command is provided by *System V* to re-install the *mm*(1) macros without these specific features. A manual page for *non-btl*(1) is provided in *Appendix A*.

7.9 Games

The set of games provided in the **/usr/games** directory is not consistent with those described in the *UNIX System User's Manual*, nor is the available set of games identical across processor lines. Table 8 attempts to clarify this confusion. It identifies the games available on each processor line, the existence of a source file, and the availability of a manual page. For each game whose source exists, the executable game was built from this source.

The **/usr/games** directory also contains a subdirectory called *lib* and two commands (*turnon* and *turnoff*) to enable and disable the use of the games. Note there is no longer a *login* entry for games in the distributed **/etc/passwd** file.

TABLE 8. State of /usr/games

Game	Source?	Manpage?	PDP?	VAX?
adventure	no	no	yes	yes
arithmetic	yes	yes	yes	yes
back	yes	yes	yes	yes
biorhythm	no	no	yes	no
bj	yes	yes	yes	yes
checkers	no	no	yes	no
chess*	no	yes	yes	no
craps	yes	yes	yes	yes
cubic	no	yes†	yes	no
fish	yes	no	yes	yes
fortune	yes	no	yes	yes
hangman	yes	yes	yes	yes
jotto	no	yes	yes	yes
mastermind	yes	no	yes	yes
maze	yes	yes	yes	yes
monopoly	no	no	yes	no
moo	yes	yes	yes	yes
number*	yes	no	yes	yes
psych*	yes	no	yes	yes
quiz	yes	yes	yes	yes
random*	yes	no	yes	yes
reversi	no	yes	yes	no
sky	no	yes	no	no
snake*	no	no	yes	yes
trk*	yes	no	yes	yes
ttt	yes	yes	yes	yes
wump	yes	yes	yes	yes

* Needs instructions.

† Not in *System V* manual; on-line page only.

8. DISCLAIMERS

This document includes specific reference to the use of the UNIX System on a particular processor, the Western Electric Company 3B20S, which is not presently available except for internal use within the Bell System. Some commands, system calls, subroutines, file formats, and special files described in the *System V* documentation and associated with the 3B20S, are not included with the release of *System V*. Manual pages associated with them are marked with *(3B20S only)* at the top of the page.

Additional commands, subroutines, and special files described in the *System V* documentation are not included with the release of *System V*. Manual pages associated with them are included in the list:

nscloop(1M)	*nscmon*(1M)	*nscstat*(1C)	*nsctorje*(1C)
nusend(1C)	*scat*(1)	*se*(1)	*stgetty*(1M)
st(1M)	*ststat*(1)	*stlogin*(1)	*trouble*(1)
x25lnk(1C)	*x25pvc*(1C)		
x25alnk(3C)	*dial*(3C)	*x25clnk*(3C)	*x25dlnk*(3C)
x25hlnk(3C)	*x25ilnk*(3C)	*x25ipvc*(3C)	*x25rpvc*(3C)
nsc(7)	*st*(7)	*stermio*(7)	*x25*(7)

Some system maintenance procedures included in the *System V* documentation are not intended for use with *System V*. Manual pages associated with them include *trouble*(8), and other manual pages, marked *(3B20S only)*, in *Section 8* of the *UNIX System Administrator's Manual*.

APPENDIX A

New Manual Pages for the
UNIX™ System User's Manual - System V
and the
UNIX™ System Administrator's Manual - System V

NAME

ex — text editor

SYNOPSIS

ex [−] [−v] [−t *tag*] [−r] [+*command*] [−l] [−x] name ...

DESCRIPTION

Ex is the root of a family of editors: *ex* and *vi*. *Ex* is a line oriented editor which is a superset of *ed*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

DOCUMENTATION

The *Ex Reference Manual* is a comprehensive and complete manual for the command mode features of *ex*, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed;* the material in the introductory and advanced documents works also with *ex*.

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi*. The *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction*.

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base *termcap*(5) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1). There is also an interline editing **open** (**o**) command which works on all terminals.

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you don't accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, filenames in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming filenames and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command & in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join** (j) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

−	Suppress all interactive-user feedback. This is useful in processing editor scripts.
−v	Invokes *vi*
−t*tag*	Edit the file containing the *tag* and position the editor at its definition.
−r*file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
+*command*	Begin editing by executing the specified editor search or positioning *command*.
−l	LISP mode; indents appropriately for lisp code, the () {} [[and]] commands in *vi* and *open* are modified to have meaning for *lisp* .
−x	Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

The *name* argument indicates files to be edited.

Ex States

Command	Normal and initial state. Input prompted for by :. Your kill character cancels partial command.
Insert	Entered by **a i** and **c**. Arbitrary text may be entered. Insert is normaiiy terminated by line having only . on it, or abnormally with an interrupt.
Open/visual	Entered by **open** or **vi**, terminates with **Q** or ^\.

Ex command names and abbreviarions

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**	open	**o**	unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	*window*	**z**
insert	**i**	rewind	**rew**	*escape*	**!**
join	**j**	set	**se**	*lshift*	**<**
list	**l**	shell	**sh**	*print next*	CR
map		source	**so**	*resubst*	**&**
mark	**ma**	stop	**st**	*rshift*	**>**

move	**m**	substitute	**s**	*scroll*	**^D**

Ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
−	previous	*'x*	marked with *x*
+*n*	*n* forward	*''*	previous context
%	1,$		

Initializing options

EXINIT	environmental variable for options
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x***?**	show value of option *x*

Useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp		( ) { } are s-exp's
list		print ^I for tab, $ at end
magic		. [* special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any char in *str*
[↑*str*]	... not in *str*
[*x*−*y*]	... between *x* and *y*
*	any number of preceding

AUTHOR

The *vi (ex)* editor is based on software developed by The Unviersity of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), vi(1), termcap(5)

CAVEATS AND BUGS

The version of *ex* that runs on the **PDP11** does not support the full command set due to space limitations. The commands which are not supported are detailed in the "Ex Reference Manual." The most notable commands which are missing are the macro and abbreviation facilities.

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line '**−**' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

NAME

non-btl — re-install *mm* macros without Bell Labs specific features

SYNOPSIS

sh non-btl.sh

DESCRIPTION

The *non-btl.sh* command will modify and re-install the source for the Memorandum Macros (used with *nroff*(1) and *troff*(1)) when Bell Laboratories specific macros are not desired.

Specifically, use of the *non-btl.sh* command will remove the **.TM** , **.PM** , **and .CS** macros, and the **}2** string (which normally contains the name *Bell Laboratories)* from the macro package. After execution of *non-btl.sh* , use of these features will have no effect.

This command does not remove the source for these features from the macro file, but does erase their definition. Those users who wish to tailor the macro package to their own environment may choose to not run *non-btl.sh* , but to modify the definition of the affected macros and string to their own specifications. Remember to re-install the macros after they are modified.

IMPORTANT

The *non-btl.sh* command is located in the */usr/src/cmd/text/macros.d* directory, and may only be used by the super-user.

NAME

fsba — file system block analyzer

SYNOPSIS

fsba file-system ...

DESCRIPTION

Fsba determines the number of extra sectors (1 sector has 512 bytes) needed when the file system logical block size is increased from 512 bytes per block to 1024 bytes/block. *File-system* should be specified by device name (e.g., **/dev/rp11**).

Fsba determines how many sectors are currently allocated for the 512 bytes/block file system, and how many sectors will be required for the 1024 bytes/block converted file system. *Fsba* also prints out the number of allocated and free i-nodes for each *file system*.

If the number of free sectors for the 1024 bytes/block file system is negative, this indicates the file-system is too large to convert to 1024 bytes/block.

SEE ALSO

fs(4).

NAME

vi — screen oriented (visual) display editor based on ex

SYNOPSIS

vi [**−t** *tag*] [**−r** *file*] [**+***command*] [**−l**] [**−w***n*] [**−x**] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi* changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file. The *Vi Quick Reference* card and the *Introduction to Display Editing with Vi* provide full details on using *vi*.

INVOCATION

The following invocation options are interpreted by *vi*:

−t*tag*	Edit the file containing the *tag* and position the editor at its definition.
−r*file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
+*command*	Begin editing by executing the specified editor search or positioning *command* .
−l	**LISP** mode; indents appropriately for lisp code, the () {} [[and]] commands in *vi* and *open* are modified to have meaning for *lisp* .
−w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
−x	Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

The *name* argument indicates files to be edited.

"VI STATES"

Command	Normal and initial state. Other states return to command state upon completion. ESC (escape) is used to cancel a partial command.
Insert	Entered by **a i A I o O c C s S R**. Arbitrary text may then be entered. Insert is normally terminated with ESC character, or abnormally with interrupt.
Last line	Reading input for **:** **/** **?** or **!**; terminate with ESC or CR to execute, interrupt to cancel.

COMMANDS

Counts before vi commands

line/column number	**z G |**
scroll amount	**^D ^U**
replicate insert	**a i A I**
repeat effect	most of the rest

Sample commands

dw	delete a word
de	... leaving white space
dd	delete a line
3dd	... 3 lines
i*text*ESC	insert text *abc*
cw*new*ESC	change word to *new*
ea*s*ESC	pluralize word
xp	transpose characters
ZZ	exit vi

Interrupting, canceling

ESC	end insert or incomplete cmd
^?	(delete or rubout) interrupts
^L	reprint screen if ^? scrambles it

File manipulation

:w	write back changes
:wq	write and quit
:q	quit
:q!	quit, discard changes
:e *name*	edit file *name*
:e!	reedit, discard changes
:e + *name*	edit, starting at end
:e +*n*	edit starting at line *n*
:e #	edit alternate file
^	synonym for :e #
:w *name*	write file *name*
:w! *name*	overwrite file *name*
:sh	run shell, then return
:!*cmd*	run *cmd*, then return
:n	edit next file in arglist
:n *args*	specify new arglist
:f	show current file and line
^G	synonym for :f
:ta *tag*	to tag file entry *tag*
^]	:ta, following word is *tag*

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	goto line (end default)
/*pat*	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/*pat*/+*n*	n'th line after *pat*
?*pat*?−*n*	n'th line before *pat*
]]	next section/function
[[	previous section/function
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z−	... at bottom
z.	... at center
/*pat*/z−	*pat* line at bottom
z*n*.	use *n* line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

``	previous context
''	... at first non-white in line
m*x*	mark position with letter *x*
`*x*	to mark *x*
'*x*	... at first non-white in line

Line positioning

H	home window line
L	last window line
M	middle window line
+	next line, at first non-white
−	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character positioning

^	first non white
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	f backward
t*x*	upto *x* forward
T*x*	back upto *x*
;	repeat last f F t or T
,	inverse of ;
|	to specified column
%	find matching ({) or }

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

Commands for LISP Mode

)	Forward s-expression
}	... but don't stop at atoms
(	Back s-expression
{	... but don't stop at atoms

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as ^H

kill	your kill, erase input this line
\	escapes ^H, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R	replace characters

Operators (double to affect lines)

d	delete
c	change
<	left shift
>	right shift
!	filter through command
=	indent for LISP
y	yank lines to buffer

Miscellaneous operations

C	change rest of line
D	delete rest of line
s	substitute chars
S	substitute lines
J	join lines
x	delete characters
X	... before cursor
Y	yank lines

Yank and put

p	put back lines
P	put before
"*x*p	put from buffer *x*
"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, redo, retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d*p	retrieve *d*'th last delete

AUTHOR

The *vi* (*ex*) editor is based on software developed by The Unviersity of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

SEE ALSO

ex (1). "Vi Quick Reference" card and "An Introduction to Display Editing with Vi", in the UNIX System Document Processing Guide.

CAVEATS AND BUGS

The version of *vi* that runs on the **PDP-11** does not support the full command set due to space limitations. The commands which are not supported are detailed in "An Introduction to Display Editing with Vi". The most notable commands which are missing are the macro and abbreviation facilities.

Software tabs using **^T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a : escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open*.

NAME

gettydefs — speed and terminal settings used by getty

DESCRIPTION

The **/etc/gettydefs** file contains information used by *getty*(1M) (see the *UNIX System Administrator's Manual*) to set up the speed and terminal settings for a line. It supplies information on what the *login* prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a <*break*> character.

Each entry in **/etc/gettydefs** has the following format:

label# initial-flags # final-flags # login-prompt #next-label

Each entry is followed by a blank line. Lines that begin with # are ignored and may be used to comment the file. The various fields can contain quoted characters of the form **\b**, **\n**, **\c**, etc., as well as *\nnn*, where *nnn* is the octal value of the desired character. The various fields are:

label This is the string against which *getty* tries to match its second argument. It is often the speed, such as **1200**, at which the terminal is supposed to run, but it needn't be (see below).

initial-flags These flags are the initial *ioctl*(2) settings to which the terminal is to be set if a terminal type is not specified to *getty*. *Getty* understands the symbolic names specified in **/usr/include/sys/termio.h** (see *termio*(7) in the *UNIX System Administrator's Manual*). Normally only the speed flag is required in the *initial-flags*. *Getty* automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until *getty* executes *login*(1).

final-flags These flags take the same values as the *initial-flags* and are set just prior to *getty* executes *login*. The speed flag is again required. The composite flag **SANE** takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.

login-prompt This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or new-line), they are included in the *login-prompt* field.

next-label This indicates the next *label* of the entry in the table that *getty* should use if the user types a <*break*> or the input cannot be read. Usually, a series of speeds are linked together in this fashion, into a closed set. For instance, **2400** linked to **1200**, which in turn is linked to **300**, which finally is linked to **2400**.

If *getty* is called without a second argument, then the first entry of **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. It is also used if *getty* can't find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into the command which will bring up a terminal at **300** baud.

It is strongly recommended that after making or modifying **/etc/gettydefs**, it be run through *getty* with the check option to be sure there are no errors.

FILES

/etc/gettydefs

SEE ALSO

getty(1M), termio(7) in the *UNIX System Administrator's Manual*.
login(1), ioctl(2).

NAME

termcap — terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals, used, *e.g.*, by *vi*(1). Terminals are described in *termcap* by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap*.

Entries in *termcap* consist of a number of ':' separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by '|' characters. The first name is always 2 characters long and is used by older systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

CAPABILITIES

(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not ^H
bs	bool		Terminal can backspace with ^H
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give :ei=: if ic
eo	str		Can erase overstrikes with a blank

ff	str	(P*)	Hardcopy terminal page eject (default ^L)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give :im=: if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by other function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of keypad transmit mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of other keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in keypad transmit mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on other function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default \n)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode

vb	str	Visible bell (may not move cursor)
ve	str	Sequence to end open/visual mode
vs	str	Sequence to start open/visual mode
xb	bool	Beehive (f1=escape, f2=ctrl C)
xn	bool	A newline is ignored after a wrap (Concept)
xr	bool	Return acts like **ce** \r \n (Delta Data)
xs	bool	Standout not erased by writing over it (HP 264?)
xt	bool	Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept−100, is among the more complex entries in the *termcap* file as of this writing. (This particular concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+
        :co#80:\:dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in
        :ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and that empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in *termcap* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has automatic margins (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co** which indicates the number of columns the terminal has gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a : in a capability it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability it must be encoded as **\200**. The routines which deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it or bugs in *ex*. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in */etc/termcap*. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor.

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H** in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on, i.e. **am**.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability, with *printf*(3s) like escapes **%x** in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the % encodings have the following meanings:

%d	as in *printf*, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output.
%r	reverses order of line and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output.
%D	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is cm=6\E&%r%2c%2Y. The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, cm=^T%.%.. Terminals which use %. need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** ^D and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cm=\E=%+ %+ .

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not not falling into one of these two

classes.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as **ei** the sequence to leave insert mode (give this, with an empty value also if you gave **im**). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining — half bright is not usually an acceptable standout mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **ug** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this

applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, :ko=cl,ll,sf,sb:, which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, then this can be given as **ta**.

Hazeltine terminals, which don't allow '~' characters to be printed should indicate **hz**. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc**. Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **x***x*.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is */usr/lib/tabset/std* but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since *termlib* routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with **xx@** where **xx** is the capability. For example, the entry

hn|2621nl:ks@:ke@:tc=2621:

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

AUTHOR

Termcap is based on software developed by The Unviersity of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/etc/termcap file containing terminal descriptions

SEE ALSO

ex(1), vi(1).

CAVEATS AND BUGS

Note *termcap* will be replaced by *terminfo* in the next release. Transition tools will be provided. *Ex* allows only 256 characters for string capabilities, and the routines in *termcap*(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

APPENDIX B

Setting Up the UNIX™ System (DEC)

Setting up the UNIX™ System (DEC)

GENERAL

This section describes the load, update, and configuration procedures involved in installing a UNIX[1] operating system on the following DEC processors:

- Digital Equipment Corp. VAX 11/780, 11/750
- Digital Equipment Corp. PDP 11/70

A. Prerequisites

Before attempting to generate a UNIX operating system, the system administrator should understand that a considerable knowledge of the related documentation is required and assumed. In particular, the administrator should be very familiar with the following documents:

- *UNIX System User's Guide*
- *UNIX System User's Manual*
- *UNIX System Administrator's Manual*
- *UNIX System Operator's Guide*
- All sections in the *UNIX System Administrator's Guide.*

Throughout this section, each reference of the form **name**(1M), **name**(7), or **name**(8) refers to entries in the *UNIX System Administrator's Manual.* All other references to entries of the form **name**(N), where "N" is a number(1 through 6) possibly followed by a letter, refer to entry **name** in section N of the *UNIX System User's Manual* .

The system administrator must have a basic understanding of the operation of the hardware. This includes the operation of the console and the tape and disk drives, which are assumed to have standard UNIX system addresses and interrupt vectors. It is also assumed that the hardware works and has been completely installed. All appropriate DEC diagnostics should have been run to test the configuration, and a detailed description of the hardware including device addresses, interrupt vectors, and bus levels is needed. This information is necessary to generate a UNIX system.

B. Procedure

The UNIX system is distributed on two magnetic tapes, recorded in 9-track format at either 800 bpi or 1600 bpi. Distribution tapes will be marked either "PDP-11" or "VAX", 800 bpi or 1600 bpi. Make sure you have the correct tape for your machine. UNIX System V is to be installed using only the *initial load procedures* described in this document.

Initial Load

The initial load program (on Tape 1) will copy a file system from tape (VAX-11/780: either a TE16 or a TU78; VAX-11/750: either a TS11 or a TU77; PDP-11: either a TU10 or a TU16) to disk (VAX: either a RP06, a RP07, a RM05, or a RM80; PDP-11: either an RP03, an RP06, or an RM05). In this document, RP04/5 drives are considered to be equivalent to RP06 drives; any differences will be noted explicitly. Once the *root* file system has been successfully loaded to disk, the UNIX system may be booted and the available utility programs used to complete the installation.

Update

The update procedures are to be used *only* on releases that are designated as *update* releases. The **cpio**(1) program is used to perform all updates. The **cpio** program will not update any file if its replacement has a modification time that is less than (i.e., earlier than) the modification time of the

1. UNIX is a trademark of Bell Telephone Laboratories, Incorporated.

original file. Certain administrative files (e.g., */etc/passwd*) are sent with a modification time of January 1, 1970 to ensure that they do not replace their counterparts during updates. Any file not copied will cause **cpio** to print a message to that effect. These messages should always be investigated to ensure that any files not copied were of that type. However, note that, depending on respective modification times, a locally-modified file may get updated, thus destroying the local modifications.

One of the most common problems that can arise during an installation is running out of disk space when performing an update. Should this occur, the original contents of the file system should be restored from a backup copy and the contents of the update tape should be read into a spare file system using the **cpio** program. Unwanted material can then be removed and the original file system can be updated from this new file system using the **−pdm** options of **cpio.**

LOAD PROCEDURES

A. Distribution Tape Format

Tape 1 contains eight files: a loader, a physical copy of the *root* file system, the **cpio** program, a **cpio** structured copy of the *root* file system, and four files (**cpio** format) that represent selectable items. *Root* refers to the directory "/", which is the root of all the directory trees. The format of this tape is as follows:

file 1:	Tape boot loader − 512 bytes; Tape boot loader − 512 bytes; Initial load program − several 512-byte records;
file 2:	*root* file system (physical) − 5120-byte records (blocking factor 10);
file 3:	**cpio** program (latest version) − several 512-byte records (to be used *only* for updating an earlier UNIX system release);
file 4:	the *root* file system (structured in **cpio** format) − several 5120-byte records (to be used *only* for updating an earlier UNIX system release);
file 5:	on-line manual pages (same format as file 4);
file 6:	on-line documents (same format as file 4);
file 7:	RJE software (same format as file 4);
file 8:	graphics software (same format as file 4).

The *root* (/) file system contains the following directories:

bck:	Directory used to mount a backup file system for file restoring.
bin:	Public commands; described in Section 1 of the *UNIX System User's Manual*
dev:	Special files, all the devices on the system.
etc:	Administrative programs and tables.
lib:	Public libraries, parts of the assembler, C compiler.
mnt:	Directory used to mount a file system.
lost+found:	Directory used by **fsck**(1M) for disconnected files.
stand:	Stand-alone programs.
tmp:	Directory used for temporary files; should be cleaned at reboot.
usr:	Directory used to mount the */usr* file system.

B. Initial Load of Root

Mount Tape 1 on drive 0 and position it at the load point.

PDP-11

Boot the tape by reading either record 0 or 1 into memory starting at address 0 and start execution at address 0. This may be accomplished by using a standard DEC ROM bootstrap loader, a special ROM, or some manual procedure; see **romboot**(8), **tapeboot**(8), and **70boot**(8).

VAX

See "Installation Boot Procedures" under either **11/780 ops**(8) or **11/750 ops**(8). These entries describe initial tape booting and modification of the console floppy disk to simplify UNIX system administration.

C. Common to PDP-11 and VAX

The tape boot loader will type "UNIX tape boot loader" on the console terminal and read in and execute the initial load program. The program will then type detailed instructions about the operation of the program on the console terminal. The program will ask what type of disk drive you have and which drive you plan to use for the copy. The disk controller used must be at the standard DEC address indicated by the program; however, other disk controllers on your system may be at nonstandard addresses. A formatted, ECC flag-free pack must be mounted on the drive you have indicated. If necessary, use the appropriate DEC diagnostic program to format the pack. For the VAX use **format**(1M). Note that the pack will be written on. Next, the program will ask what type of tape drive you have and which drive contains the tape. Normally, this will be drive 0, but the program will work with other drives. Note that the tape is currently positioned correctly after the end-of-file between the initial load program and the *root* file system. When everything is ready, the program will copy the file system from the tape to disk and give instructions for booting the UNIX system. After the copy is complete and you have booted the basic version of the UNIX operating system, check [using **fsck**(1M)] the *root* file system and browse through it.

PDP-11/70 Only

The file */stand/mmtest* is a stand-alone memory mapping diagnostic program for the PDP-11/70. If you are not *absolutely* sure that DEC FCO (field change order) M8140-R002 has been applied to your PDP-11/70 CPU, *stand/mmtest* should be booted and allowed to run at least 20 minutes. To boot this program, go through the disk boot procedure, but specify:

0=stand/mmtest

The UNIX system comes with optional power-fail recovery. This feature *requires* that the *power-up* and *power-down* interrupt vectors be 024.

D. Update of Root

It is very important that the system be running in *single-user* mode during the update phase. To update an already existing *root* file system, files three and four on Tape 1 will be used. It is necessary to first make a copy of the *root* file system using **volcopy**(1M) and then update this copy. The copy should be made on a separate disk pack using the same section number as the *root* file system (always section 0). Also, after the update is completed, check if any of the local administrative files in the directory */etc* need modification. Most of these are mentioned in the part "ADMINISTRATIVE FILES".

Mount Tape 1 on drive 0 and position it at the load point. It is assumed that disk drive 1 is available for making the copy and that the *root* file system is on */dev/rp0*. The following procedure will first make a copy of the *root* file system, and then update this copy. Note that */dev/mt4* refers to tape drive 0 but has the side effect of spacing forward to the next end-of-file (no rewind option). The **−B** option of **cpio** specifies that input is in 5120-byte records:

```
volcopy root /dev/rrp0 pkname1 /dev/rrp10 pkname2
mount /dev/rp10 /bck
#  The 2 echoes will move the tape to file 3
echo </dev/mt4
echo </dev/mt4
cp /dev/mt4 /bck/bin/cpio
chmod 755 /bck/bin/cpio
chown bin /bck/bin/cpio
cd /bck
/bck/bin/cpio -idmB </dev/rmt0
cd /
umount /dev/rp10
```

Pkname1 and *pkname2* are the volume names of the source and destination disk packs, respectively. If the new copy is satisfactory, shut down and halt the system, change disk packs, and reboot the system using the new *root*.

E. Tape 2 (/usr) Format

Tape 2 contains the */usr* file system in **cpio** format (5120-byte records). The */usr* file system contains commands and files that must be available (mounted) when the system is in *multiuser* mode. The tape contains the following directories:

adm:	Miscellaneous administrative command and data files including the process accounting file *pacct*.
bin:	Public commands; an overflow for */bin*.
games:	Various demonstration and instructional programs.
include:	Public C language *#include* files.
lib:	Archive libraries including the text processing macros; also contains data files for various programs such as **spell**(1) and **cron**(1M).
mail:	Mail directory.
lost+found:	Directory used by **fsck**(1M) for disconnected files.
news:	Place for all the various system news; see **news**(1).
pub:	Handy public information, e.g., table of ASCII characters.
spool:	Spool directory for daemons.
src:	Source for commands, libraries, the system, etc.
tmp:	Directory for temporary files; should be cleaned at reboot.

F. Initial Load of /usr

Mount Tape 2 on drive 0 and position it at the load point. Mount a file system (device) as */usr*. The ultimate size and location of this file system on a device is an administrative decision; initially, the following procedure will suffice:

```
mkfs /dev/rrp1 65000 gap blocks
#  See mkfs(1M) for appropriate parameters.
#  Note that for RP03 disks, this will use part of section 2.
labelit /dev/rrp1 usr pkname
mount /dev/rp1 /usr
chmod 775 /usr
cd /usr
cpio -idmB </dev/rmt0
```

Pkname is the volume name of the pack (e.g., "p0001").

Because */usr* must be mounted when the system is in *multiuser* mode, the file */etc/rc* must be changed to include the command lines to mount and unmount the file systems in *single user* and *multiuser* mode. These lines must be inserted at the appropriate places in */etc/rc* as indicated by comments in the prototype file. Next, the file */etc/checklist* should be changed to include the file system device (e.g., */dev/rrp1*); see **fsck**(1M), **labelit**(1M), **mkfs**(1M), **mount**(1M), and **checklist**(4).

G. Update of /usr

It is advisable that the system be running in *single*user mode during the update phase. It is also wise to first make a copy of your */usr* file system for backup purposes. Next, mount Tape 2 on drive 0 and position it at the load point. The */usr* file system *must* also be mounted. The following procedure will perform the update:

```
cd /usr
cpio -idmB </dev/rmt0
```

H. Initial Load or Update of Selectable Items

The initial load and update procedures are essentially the same; the only exception being the creation of the selectable item directory on the initial load.

Mount Tape 1 on drive 0 and position it at the load point. Make sure that the */usr* file system is mounted. The following procedure will read in the source for each of the selectable subsystems. If a particular subsystem is not desired, simply skip that file on the tape by executing the following command:

```
echo </dev/rmt4
```

The tape can be rewound after any subsystem by specifying */dev/rmt0* instead of */dev/rmt4*.

```
echo </dev/rmt4; echo </dev/rmt4
echo </dev/rmt4; echo </dev/rmt4

cd /usr
mkdir man; chown bin man; chgrp bin man; chmod 775 man
cd man
cpio -idmB </dev/rmt4

cd /usr
mkdir docs; chown bin docs; chgrp bin docs; chmod 775 docs
cd docs
cpio -idmB </dev/rmt4

cd /usr/src/cmd
mkdir rje; chown bin rje; chgrp bin rje; chmod 775 rje
cd rje
cpio -idmB </dev/rmt4

cd /usr/src/cmd
mkdir graf; chown bin graf; chgrp bin graf; chmod 775 graf
cd graf
cpio -idmB </dev/rmt0
```

After installing the source for the *rje* and *graphics* subsystems, the software must be *built* and *installed*. (Execute the commands for the subsystems that you have elected to take.)

To build and install *rje*, change your working directory to */usr/src* and execute one of the following **:mkcmd** lines:

```
ARGS="rje1"    ./:mkcmd rje    # makes a single IBM system
ARGS="rje2"    ./:mkcmd rje    # makes rje1, and rje2
ARGS="rje3"    ./:mkcmd rje    # makes rje1, rje2, and rje3
ARGS="rje4"    ./:mkcmd rje    # makes rje1, rje2, rje3, and rje4
```

See **rje**(8) and the "UNIX System Remote Job Entry " section for additional information.

To build and install the graphics package:

```
cd /usr/src
./:mkcmd graf
```

CONFIGURATION PLANNING

A. UNIX System Configuration

The basic UNIX operating system supplied supports only the console, a disk controller (disk drive 0), and a tape controller (tape drive 0). Each system administrator must describe the actual configuration of their own system.

All of the UNIX operating system source code and object libraries are in */usr/src/uts*. All of the configuration information is kept in the directory */usr/src/uts/*/cf*; the "*" represents either *pdp11* or *vax*. There are only two files that must be changed to reflect your system configuration - *low.s* (*univec.c* on the VAX) and *conf.c*. The program **config**(1M) should be used to make these changes.

Config requires a "system description file" and produces the two needed files. Table 4.A lists the values and sizes of the basic parameters for the different CPUs. For more details of syntax and structure, see **config**(1M) and the associated **master**(4).

The first part of the system description file lists all of the hardware devices on the system. Next, various system information is listed. A brief explanation of this information follows:

TABLE 4.A.

Item	PDP-11/70		VAX-11/750,/780	
	Range	Size	Range	Size
nswap	3000	—	9000	—
buffers	25-60	30†	80-400	56‡
sabufs	10-15	542	—	—
hashbuf	32-128	6	32-128	12
physbuf	3-5	30	3-7	56
inodes	100-250	24	100-300	84
iblocks	80-200	52	—	—
files	100-250	8	100-300	12
mounts	8-16	8	8-20	20
coremap	50-100	4	—	—
swapmap	50-100	4	50-100	4
calls	30-60	6	30-60	12
procs	50-200	32	50-200	64
texts	25-50	12	25-50	16
clists	100-300	28	100-250	72
maxproc	25	—	25	—

† Plus 512 bytes/buffer outside system space.

‡ Plus 1024 bytes/buffer allocated at start up.

- *root*—— Specifies the device where the *root* file system is to be found. The device must be a block device with read/write capability because this device will be mounted read/write as "/". Thus, a tape can not be mounted as the *root* but can be mounted as some read-only file system. Normally, *root* is disk drive 0, section 0.

- *pipe*—— Specifies where pipes are to be allocated (must be a *mounted* file system — the root file system is normally used).

- *dump*—— Specifies the device to be used to dump memory after a system crash. Currently, only the TU10, TU16/TE16, TU78, and TS11 tape drives are supported for this purpose.

- *swap*—— Specifies the device and blocks that will be used for *swapping*. *Swplo* is the first block number used and *nswap* indicates how many blocks, starting at *swplo*, to use. Care must be taken that the swap area specified does not overlap any file system. For example, if section 0 is 8000 blocks long, the *root* file system could occupy the first 6000 blocks, and *swap* the remaining 2000 by specifying:

```
root rp06 0
swap rp06 0 6000 2000
```

- *buffers*——Specifies how many "system buffers" to allocate. Real-time response improves as more buffers are allocated. UNIX system buffers form a "data cache". Improvement in the hit rate of this cache tends to fall as the number of buffers is increased.

- *sabufs*——PDP-11 only: specifies how many "system addressable" buffers to allocate. One buffer is needed for every mounted file system. Certain I/O drivers need such buffers.

- *hashbuf*——Specifies how many hash buckets to allocate. These are used to search for a buffer given a device number and block number. This number must be a power of two. The default value is 64.

- *physbuf*——Specifies how many physical I/O buffer headers to allocate. One is needed for each physical read or write active. The default value is 4.

- *inodes*——Specifies how many "inode table" entries to allocate. Each entry represents a unique open inode. When the table overflows, the warning message "Inode table overflow" will be printed on the console. The table size should be increased if this happens regularly. The number of entries used depends on the number of active processes, texts, and mounts.
- *iblocks*——PDP-11 only: specifies how many "inode block address cache" entries to allocate. An entry is needed for each regular, directory, or fifo file that is open. Since special files do not need an entry, data space can be conserved by specifying a much smaller number of iblocks than inodes. The default value is four less than the number of inodes.
- *files*——Specifies how many "open-file table" entries to allocate. Each entry represents an open file. When the table overflows, the warning message "no file" will be printed on the console. The table size should be increased if this happens regularly.
- *mounts*——Specifies how many "mount table" entries to allocate. Each entry represents a mounted file system. The *root* (/) will always be the first entry. When full, the **mount**(2) system call will return the error EBUSY.
- *coremap*——PDP-11 only: specifies how many entries to allocate to the "list of free memory". Each entry represents a contiguous group of 64-byte blocks of free memory. When the list overflows, due to excessive fragmentation, the system will print a warning message on the console. This condition results in the loss of memory being freed. The system should be remade with a larger table size and rebooted. The number of entries used depends on the number of processes active, their sizes, and the amount of memory available.
- *swapmap*——Specifies how many entries to allocate to the "list of free swap blocks". Exactly like the *coremap*, except it represents free blocks in the swap area, in 512-byte units.
- *calls*——Specifies how many "call-out table" entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler. The time unit is 1/60 of a second. The call-out table is used by the terminal handlers to provide terminal delays and by various other I/O handlers. When the table overflows, the system will crash and print the panic message "Timeout table overflow" on the console. This value must be greater than two.
- *procs*——Specifies how many "process table" entries to allocate. Each entry represents an active process. The scheduler is always the first entry and **init**(1M) is always the second entry. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. The average number of processes per user is in the range of 2 through 5. When full, the **fork**(2) system call will return the error EAGAIN.
- *texts*——Specifies how many "text table" entries to allocate. Each entry represents an active read-only text segment. Such programs are created by using the **−i** or **−n** option of the loader **ld**(1). The **−n** option is implicit on the VAX. When the table overflows, the warning message "out of text" is printed on the console.
- *clists*——Specifies how many "character list buffers" to allocate. On the PDP-11, each buffer contains up to 24 bytes; on the VAX, each buffer contains up to 64 bytes. The buffers are dynamically linked together to form input and output queues for the terminal lines and various other slow-speed devices. The average number of buffers needed per terminal line is in the range of 5 through 10. When full, input characters from terminals will be lost and not echoed.
- *maxproc*——Specifies how many concurrent processes a non-superuser is allowed to run.
- *power*——Specifies whether to attempt restart after a power failure. A value of 0 (default) indicates no restart; a value of 1 attempts power-fail restart. On restart, device drivers are called and process 1 (i.e., **init**) is sent a hangup signal; see **init**(1M).

- *sema*--Specifies whether to include semaphore code. A value of 0 (default) indicates no semaphores; a value of 1 includes semaphores.
- *shmem*--Specifies whether to include shared memory code. A value of 0 (default) indicates no shared memory; a value of 1 includes shared memory.
- *mesg*--Specifies whether to include message code. A value of 0 (default) indicates no messages; a value of 1 includes messages.
- *shmmax*--Specifies the maximum size of a shared memory segment.
- *shmmin*--Specifies the minimum size of a shared memory segment.
- *shmmni*--Specifies the maximum number of shared memory segments in the system.
- *shmseg*--Specifies the maximum number of shared memory segments a user may have attached.
- *shmall*--Specifies the maximum amount of shared memory that may be allocated system wide. The default value is 512 clicks, 250k bytes.
- *shmbrk*--Specifies the number of clicks between the end of the data segment, and the beginning of the first shared memory segment if the default starting address is used allowing the user to continue to use **sbrk**(2) or **brk**(2). The default value is 16 clicks, 8k bytes.
- *msgmax*--Specifies the maximum message size.
- *msgmnb*--Specifies the maximum number of bytes on any one queue.
- *msgtql*--Specifies the number of system message headers, i.e., maximum number of outstanding messages.
- *msgssz*--Specifies the message segment size. Messages consist of a set of contiguous message segments large enough to fit the text. The segments are used to help eliminate fragmentation and speed message buffer allocation. A message may span several segments.
- *msgseg*--Specifies the number of message segments in the system.
- *msgmap*--Specifies the message segment map size.
- *msgmni*--Specifies the maximum number of message queues system wide. The default is 10.
- *semmap*--Specifies the number of entries in the semaphore map. The map is used by the system to allocate and free semaphore sets. This parameter should be changed to reflect changes in *semmns*.
- *semmni*--Specifies the number of semaphore identifiers, i.e., number of semaphore sets.
- *semmns*--Specifies the number of semaphores in the system.
- *semmnu*--Specifies the number of undo structures in the system.
- *semume*--Specifies the maximum number of undo entries per structure.
- *semmsl*--Specifies the maximum number of semaphores per semaphore identifier.
- *semopm*--Specifies the maximum number of semaphore operations per **semop**(2) call.
- *maus*--PDP-11 only: specifies whether to include maus (shared memory) code. A value of 0 (default) indicates no shared memory; a value of 1 includes the shared memory code.

B. UNIX System Generation

Before generating the *first* UNIX system, the system administrator should modify the file called *Makefile* in the */usr/src/uts/*/cf* directory. This file contains five symbols that are used for system identification; it is used to initialize the internal *utsname* structure [see **uname**(1) and **uname**(2)].

The five symbols (which are eight characters maximum) are as follows:

SYS	System name (e.g., *pwba*);
NODE	The name by which the system is known on the **uucp**(1C) network (e.g., *pwba*);
REL	The operating system release (e.g., *sysV*);
VER	The current version of the system; this is usually four characters indicating when the system was made (e.g., *0620* for June 20);
MACH	The machine hardware name (e.g., *vax-780*).

Generally, only the first two symbols need local modification; the REL symbol is a constant for the duration of this release, while the VER symbol will be defined when you **make**(1) the system. The name of the executable file produced by the generation procedure will be the concatenation of the SYS and VER symbols (e.g., *pwba0620*).

To generate a new UNIX operating system, follow the procedure given:

```
cd /usr/src/uts/*/cf
ed dfile
■
  [information as described above]
.
w
q
config dfile
make VER=ver
```

The PDP-11 system has a relatively small address space. If the table sizes or the number of device types are too large, the program **sysfix** will print various error messages and the above procedure will only create an *a.out* file. In particular, the maximum available data space is 49,152 bytes. The actual data space requested can be found by using **size**(1) on *a.out* and adding the *data* and *bss* segment sizes. One then reduces the specified values for the various system entries until it all fits. The amount of space in the *bss* segment used for each entry is indicated in the part "UNIX System Configuration".

On the VAX, the combined data space should not exceed 200,000 bytes.

The PDP-11 system is distributed with a special *overlay loader* that allows larger systems (text size greater than 64K) to be configured. This system will be made up of one main segment and seven overlay text segments that use *supervisor* registers to switch the overlay text segments. The main segment may be as large as 56K, and the overlay text segments may not be greater than 8K each. To invoke this new text scheme, edit */usr/src/uts/*/Makefile*. To change the TYPE symbol from *id* to *ov* or when generating a new system, use the following procedure:

```
cd /usr/src/uts/*/cf
make VER=ver TYPE=ov
```

When you are satisfied with the new system, test it by the following procedure:

```
cp /usr/src/uts/*/sysver /   # sysver as above
cd /
umount /dev/rp1
rm /unix
ln /sysver /unix
sync
```

Halt the processor and reboot the system. Note that this procedure results in two names for the operating system object the generic */unix* and the actual name, say */pwba0620*. An old system may

be booted by referring to the actual name, but remember that many programs [such as **ps**(1)] use the generic name */unix* to obtain the *name-list* of the system.

If the new system does not work, verify that the correct device addresses and interrupt vectors have been specified. On the PDP-11, if the *wrong* interrupt vector and the *correct* device address have been specified for a device, the operating system will print the error message "stray interrupt at XXX" when the device is accessed, where XXX is the correct interrupt vector. On the VAX, the message is "stray UBA interrupt at XXX". On the PDP-11, if the *wrong* device address is specified, the system will crash with a panic trap of type 0 (indicating a time-out) when the device is accessed. On the VAX, the system will not crash directly but will print a "UBA" warning message indicating the failing address.

For the VAX, a stand-alone test may be executed to compare the configuration in */unix* with the actual hardware. After the stand-alone shell prompts "$$", type */stand/vcf*. The **vcf**(1M) command reports on the MASSBUS adapter configuration and attempts to determine the UNIBUS system device configuration. Any errors should be corrected before continuing.

C. Special Files

A special file must be made for every device on your system. Normally, all special files are located in the directory */dev*. Initially, this directory will contain:

console	console terminal
error	see **err**(7)
mem, kmem, null	see **mem**(7)
tty	see **tty**(7)
rp[0-7], rrp[0-7]	disk drive 0, sections 0-7
rl[0-1], rrl[0-1]	disk drives 0 and 1
rk[0-1], rrk[0-1]	disk drives 0 and 1
mt0, rmt0	tape drive 0
mt4, rmt4	tape drive 0 (no rewind).

These special files are of two types - block and character. This is indicated by the character *b* or *c* in the listing produced by **ls**(1) with the **−l** option.

In addition, each special file has a major device number and a minor device number. The major device number refers to the device type and is used as an index into either the *bdevsw* or *cdevsw* table in the configuration file *conf.c* The minor device number refers to a particular unit of the device type and is used only by the driver for that type. The **config** program with the **−t** option will list major device numbers.

The program **mknod**(1M) creates special files. For example, the following would create *part* of the initially-supplied */dev* directory (on the PDP-11):

```
cd /dev
mknod console c 0 0
mknod error c 20 0
mknod mem c 2 0; mknod kmem c 2 1; mknod null c 2 2
mknod tty c 13 0
mknod rp0 b 0 0; mknod rrp0 c 7 0
mknod mt0 b 1 0; mknod rmt0 c 6 0
mknod mt4 b 1 4; mknod rmt4 c 6 4
```

After the special files have been made, their access modes should be changed to appropriate values by **chmod**(1). For example:

```
cd /dev
chmod 622 console
chmod 444 error
chmod 440 mem kmem
chmod 666 null
chmod 666 tty
chmod 400 rp0 rrp0
chmod 666 mt0 rmt0
chmod 666 mt4 rmt4
```

Note that file names have no meaning to the *operating system* itself; only the major and minor device numbers are important. However, many *programs* expect that a particular file is a certain device. Thus, by convention, special files are named as follows:

block device	*conf.c*	*/dev*
RP03 disk	rp	rp*
RP04/5/6 disk	hp	rp*
RS03/4 fixed head disk	hs	rs*
RK05 disk	rk	rk*
RL01/2 disk	rl	rl*
RM05 disk	hm	rp*
RM80 disk	gd	rp*
RP07 disk	gd	rp*
general disk	gd	rp*
TU10 tape	tm	mt*
TE/TU16 tape	ht	mt*
TS11 tape	ts	mt*
TU78 tape	ts	mt*
general tape	gt	mt*

character device	*conf.c*	*/dev*
DL11 async. line	kl	tty*, console
DH11 async. line mux	dh	tty*
DMC11 sync. unit	dmc	dmc*
DZ11 async. line mux	dz	tty*
DN11 auto call unit	dn	dn*
DU11 sync. line	du	du*
KMC11-B micro	kmc	kmc*
DM11-BA modem control	dmk	dmk*
DZ11/KMC11-B assist	dzb	tty*
LP11 line printer	lp	lp*
RP03 disk	rp	rrp*
RP04/5/6 disk	hp	rrp*
RM05 disk	hm	rrp*
RM80 disk	gt	rrp*
RP07 disk	gt	rrp*
general disk	gd	rrp*
RS03/4 fixed head disk	hs	rrs*
TU10 tape	tm	rmt*
TE/TU16 tape	ht	rmt*
TS11 tape	ts	rmt*
TU78 tape	gt	rmt*
general tape	gt	rmt*
error	err	error
memory	mm	mem, kmem, null
terminal	sy	tty

For those devices with a */dev* name ending in "*", this character is replaced by a string of digits representing the *minor* device number. For example:

```
mknod /dev/mt1 b 1 1
mknod /dev/rp24 b 0 024 # leading zero means octal
mknod /dev/tty03 c 1 3
```

Note that for disks, an octal number scheme is maintained because each drive is logically partitioned into eight sections. Thus, */dev/rp24* refers to section 4 of physical drive 2. There is also a special file, */dev/swap*, that is used by the program **ps**(1). This file must reflect what *block* device is used for swapping and must be readable. For example:

```
rm /dev/swap
mknod /dev/swap b 0 0
chmod 440 /dev/swap
chown sys /dev/swap; chgrp sys /dev/swap
```

The minor device numbers for tapes are also encoded; the minor device number consists of the four bits shown in Fig. 4.1.

DENSITY SELECT	0=REWIND 1=NO REWIND	DRIVE-SELECT BIT 1	DRIVE-SELECT BIT 0

Minor Device Number, Tape

FIG. 4.1 -

For the TU10 and TE16, the density select bit is 0 for 800 bpi and 1 for 1600 bpi. For the TU78, the density select bit is 0 for 1600 bpi and 1 for 6250 bpi. For the TS11, the density select bit is ignored; the density is always 1600 bpi. Therefore, the special file for tape drive 1, for

operation at 1600-bpi, with no rewind on close would have a minor device number of 13 (binary 1101). The 2-bit drive number field permits a maximum of four drives per system.

The minor numbers of the automatic calling unit (ACU) interface, DN11, are encoded in eight bits as shown in Fig. 4.2.

SHARED-ACU-SELECT FOUR BITS	SYSTEM-UNIT-SELECT DN11-AA TWO BITS	LINE-UNIT-SELECT DN11-DA TWO BITS

Minor Device Number, DN11

FIG. 4.2 -

The *shared-ACU-select* field in normally zero, which indicates the standard 801 ACU hardware (one ACU, one phone line). For the new shared ACU hardware (one ACU, several phone lines), the *shared-ACU-select* field indicates which line is to be used.

D. File Systems

Each physical pack is partitioned into eight logical sections. This partitioning is defined in the operating system by a table with eight entries. Each table entry is two words long. The first specifies how many blocks are in the section; the second specifies the starting cylinder; see **hp**(7) (RP04/5/6), **hm**(7) (RM05), **rp07**(7) (RP07), **rm80**(7) (RM80), and **rp**(7) (RP03) for default cylinder and block assignments. These values are described to the system in the header file */usr/include/sys/io.h*.

A file system starts at block 0 of a section of the disk and may be as large as the size of that section; if it is smaller than the size of a section, the remainder of that section is unused. Note that the sections themselves may overlap physical areas of the pack, but the file systems must *never* overlap.

The program **mkfs**(1M) (for 1K byte/block file systems) or **omkfs** (for 512 byte/block file systems) is used to initialize a section of the disk to be a file system. The length of each section of the disk is specified in 512 byte sectors. When **mkfs** is used, it produces half the number of 1K byte file system blocks. The 1K byte blocks provide better throughput for the particular file system. A 512 byte/block file system can be made using **omkfs** in place of **mkfs**. The number of physical disk sectors (512 bytes each) used to make the entire file system would be the same for either command. In future releases the functions of **mkfs** and **omkfs** will be merged into one command.

Next, the program **labelit**(1M) is used to label the file system with its name and the name of the pack. Finally, the file system may be checked for consistency by using **fsck**(1M). The file system may then be mounted using **mount**(1M).

E. DZ11 Software with KMC11-B Assist

KMC11-B microprocessors may be used to control DZ11 asynchronous multiplexers, thus off-loading terminal-oriented functions from the CPU. The KMC11-B does DMA input and output of data, character translations, tab expansions, etc. Each KMC11 can control up to four DZ11 multiplexers for a total of 32 asynchronous lines. Each system can support up to four KMC11 microprocessors; however, only 64 DZ11 lines can be controlled by KMC11 microprocessors.

Installation

1. Generate a system by including each DZ11 to be controlled by a KMC11 in the system description file. For example:

   dzb X Y Z

   where *X* is the vector address, *Y* is the UNIBUS address, and *Z* is the bus request level. Also include the KMC11 microprocessors in the configuration file:

```
kmc11    X    Y    Z
```

2. Update the files */etc/brc* and */etc/powerfail* to execute **dzbload** before entering one of the numbered *init* states and after a power failure, respectively. Each KMC used to control a DZ11 must be loaded with microcode. For each KMC11 used to control a DZ11, include

   ```
   /etc/dzbload /dev/kmc?
   ```

   where **?** is the minor device number of that KMC11.

3. Special files must be created for each KMC and DZ11 line:

   ```
   /etc/mknod   /dev/kmc?   c   X   ?
   /etc/mknod   /dev/tty??  c   Y   Z
   ```

   where *X* is the major device number of the KMC11 and **?** is the minor device number of the KMC11 controlling the DZ11 multiplexers, i.e., the KMC11 loaded with microcode in Step 2. Y is the major device number of the *dzb* device as is supplied by **config**(1M). Z is the minor device number for a particular line on a DZ11. This number is composed of three fields. The low-order three bits are the line number relative to a DZ11. The next three bits contain the unit number of the DZ11 controlling these lines. Note that this number is the absolute DZ11 number, not the number relative to the KMC. The high-order two bits are the minor device number of the KMC controlling this DZ11. For example:

   ```
   mknod /dev/tty?? c Y 0241
   ```

   specifies the second line (0 through 7) on the fifth DZ11 to be controlled by the KMC11 with minor device number 2. The DZ11 number is specified by the order of appearance in the configuration file. The first four DZ11 multiplexers must be associated with one KMC11 and the next four must be associated with another KMC11. The order in which the KMC11 microprocessors are specified is not significant.

F. Virtual Protocol Machine (VPM) Devices

KMC11-B microprocessors may be used to control DMC11-DA, -FA, or -MD single-line synchronous communications interfaces or DMS11-DA 8-line synchronous communications multiplexers (one per KMC). The combination of a KMC11 and a DMS11-DA is known as a KMS11. The KMC11 executes the link-level communications protocol and performs DMA data transfers between main memory and the communications device. A link-level protocol description in a high-level language is compiled on the host computer [see **vpmc**(1M)] and then down-loaded into the KMC11 by the **vpmstart**(1M) command.

The common synchronous interface (CSI) contains a set of utility routines for communicating with the VPM interpreter in the KMC11. The VPM protocol driver provides a transparent user interface to the VPM interpreter. The CSI routines are used not only by the VPM protocol driver but also by several other protocol drivers.

Installation

1. Generate a system. The system description file must contain a line of the form:

   ```
   vpm    0    0    0    n
   ```

   where *n* is the number of VPM protocol-driver minor devices desired, and a line of the form:

   ```
   csi    0    0    0    m
   ```

   where *m* is the number of CSI minor devices desired. One CSI minor device is required for each VPM protocol-driver minor device. If other drivers that use CSI are configured, additional CSI minor devices must be configured to support these drivers. If the VPM trace capability [see **vpm**(7)] is to be used, the system description file must also contain a line of the form:

```
trace    0    0    0    k
```

where *k* is the number of trace minor devices desired (normally one). *Vpmbsz*, a configurable parameter which gives the size of the buffer pool available to *vpm*, may be adjusted to be different from the default value found in */etc/master*. Refer to **master**(4) for more information.

2. Compile the protocol script using **vpmc**(1M):

```
vpmc [−m] [−i hdlc[/kms]] −o script.o script.r
```

where *script.r* is the source file for the protocol description. The **−m** option is used only if the protocol source requires expansion with the **m4**(1) preprocessor; otherwise, the C preprocessor is used. The **−i hdlc** option is used to compile a script that uses the bit-oriented (HDLC) version of the VPM interpreter; otherwise, the character-oriented (BISYNC) version of the interpreter will be used. If the script is to be loaded into a KMS11 8-line multiplexer rather than into a KMC11 controlling a single-line interface, the **−i hdlc/kms** option is needed. Only bit-oriented protocols are currently supported by VPM on the KMS11.

3. Install the compiled protocol description in a convenient directory such as */lib* and modify the */etc/rc* file to load the compiled protocol description into the selected KMC11s each time the system is rebooted (first transition to multiuser) using **vpmstart**(1M). If power-fail recovery is desired, the */etc/powerfail* file should also be modified to reload the compiled protocol description into the selected KMC after a power fail occurs. The **−r** option to **vpmstart** is required when reloading the KMC after a power fail.

4. Special files must be created for each VPM protocol-driver minor device, each trace minor device, and each KMC11:

```
/etc/mknod   /dev/kmc?   c   X   ?
/etc/mknod   /dev/vpm?   c   Y   ?
/etc/mknod   /dev/trace  c   Z   0
```

where *X*, *Y*, and *Z* are the major device numbers of the KMC11 driver, VPM protocol driver, and trace driver, respectively. (It is not necessary to make special files for the CSI minor devices since these are never referenced explicitly).

5. Modify the */etc/brc* file so that it executes a **vpmset**(1M) command for each VPM protocol-driver minor device. The arguments to the **vpmset** command specify the protocol-driver minor device, the KMC11, and, if the device is a KMS11, the particular line on the KMS11:

```
/etc/vpmset /dev/vpm? /dev/kmc? [n]
```

KMC11 lines are numbered starting with zero. The **vpmset** commands should be executed on the first transition to multiuser. An attempt to open a VPM protocol-driver minor device for reading and/or writing will fail if the corresponding **vpmset** command has not yet been executed.

Hardware Installation and Switch Settings

The KMC11-B microprocessor and DMC11-DA, -FA, or -MD line unit must be installed in adjacent slots of a PDP-11 or VAX backplane. Care should be taken not to exceed the dc power capacity of the cabinet in which the items are installed. The microprocessor and line unit are interconnected by a 1-foot ribbon cable. The DMC11-DA or -FA line unit is connected to a suitable synchronous modem by a DEC-supplied modem cable. If the HDLC interpreter is used, the modem must be optioned for full-duplex (4-wire) operation; at speeds above 1200 bits per second, this will normally require a private line. The DMC11-DA has an RS-232 interface that is suitable for connection to data sets such as the 208 and 209. The DMC11-FA has a CCITT V.35 interface. The DMC11-MD has an integral 56 kB modem; this unit must be connected by a pair of coaxial cables to another DMC11-MD. The device address and interrupt vector address switches on the KMC11 should be set for the selected addresses. The KMC11 should also be wired for the

selected bus priority (normally 5). All switches and jumpers on the DMC11 line unit should be in the normal configuration prescribed by the relevant DEC maintenance manual, but with one exception - the NO CRC switch (switch S2 in switch pack number 1) should be in the ON position. The purpose of this switch setting is to inhibit hardware CRC generation. Hardware CRC generation is not used with the VPM software for this device.

The procedures for installing a KMS11 are similar except that:

1. There are no switches to set on the DMS11-DA 8-line multiplexer.
2. Modem cables are not supplied with the DMS11-DA. Instead, there is an 8-line distribution panel with an RS-232 connector for each line on the DMS11-DA. Each in-use line must be connected by a suitable cable to a synchronous modem. The modem must be optioned for full-duplex (4-wire) operation. The modem should also be optioned to hold the transmitter carrier continuously on because the request-to-send lead to the modem is not controlled by the VPM software for the KMS11. The DM11-BA modem control multiplexer normally included as part of the KMS11 hardware package is not currently used by the VPM software for the KMS11. Limited access to the DM11-BA modem control unit is provided by a separate driver [see **dmk**(7)]. If the modem cannot be optioned for continuous carrier, the *dmk* driver can be used to turn on the request-to-send lead to the modem. For dial-up lines (e.g., a 212 data set operating in synchronous mode), it may be necessary to use the *dmk* driver to turn on the data-terminal ready lead to the modem.

The KMC11 contains a hardware timer which is used by both the VPM and DZ11/KMC11 software. For proper operation with this software, the value of the hardware timer should be 75 microseconds. Newer versions of the KMC11 microprocessor (etch revisions E and later) contain a switch pack at location E82 which is used to select the value of the program timer. E82-8 ON provides a time-out value of 115 *milliseconds;* E82-8 OFF yields the correct value of 75 microseconds. (E82 1 through 7 are not used). On earlier versions of the KMC11, the time-out value is selected by installing the proper value of capacitor C40. For proper operation with VPM and DZ11/KMC11 software, the value of this capacitor should be 4700 pF.

Parallel Communication Link (PCL) Devices

A *PCL* provides the interface for up to two Digital Equipment Corporation PCL11−B network buses. Each bus can be used to interconnect up to 16 CPU's, providing relatively fast communication without individual point-to-point connections.

The interface permits simultaneous bi-directional communication between any machines on a single bus. Additionally, each such path is further subdivided into 8 independent channels. A single control interface is provided to reduce the line monitoring overhead for a daemon process.

Installation

1. Generate a system. For each PCL11−B device, the system description file must contain a line of the form:

   pcl11b X Y 5 Z

   where X is the transmitter interrupt vector, Y is the UNIBUS address of the transmitter command register, and Z is the number of CPU's attached to that PCL11−B network bus.

   In addition, the PCL *control interface* must be configured into the system. This is done by adding the following line to the system description file:

   pclctrl 0 0 0 1

2. Special files must be made for each PCL minor device. Each PCL minor device specifies all information needed to connect to one of the other CPU's attached to the network bus; the minor device number is constructed as follows:

the low order 3 bits	specify a channel number.
the next 4 bits	specify one of 16 machines. (This number must be one less than the PCL11−B receiver's TDM bus address for that machine.)
The next bit	specifies one of the 2 PCL11−B's

The procedure for installing the PCL special files is as follows:

a. mkdir /dev/pcl

b. cd /dev/pcl

c. For each remote CPU attached to the PCL network bus, the following special files are made:

```
/etc/mknod   <machine name>0   c   33   X
/etc/mknod   <machine name>1   c   33   X
/etc/mknod   <machine name>2   c   33   X
/etc/mknod   <machine name>3   c   33   X
/etc/mknod   <machine name>4   c   33   X
/etc/mknod   <machine name>5   c   33   X
/etc/mknod   <machine name>6   c   33   X
/etc/mknod   <machine name>7   c   33   X
```

Where *<machine name>* is the name by which the remote machine is known on the network and X is the minor device as calculated above; the channel number for this calculation is the digit following *<machine name>*.

d. Make the special file for the PCL control interface by executing:

```
/etc/mknod   ctrl   c   34   0
```

e. Finally, give the special files the proper owner, group, and modes.

```
chown   daemon   *
chgrp   daemon   *
chmod   660      *
```

Hardware Installation and Switch Settings

The installation of the PC11−B hardware is documented in Digit al Equipment Corporation publication YC-A20TC-00, "PCL11−B Parallel Communication Link Differential TDM Bus". Because of the design of the UNIX PCL device driver, the transmitter's and receiver's TDM bus addresses must have the same value. The TDM bus addresses are also restricted to the range from 1 to 16.

ADMINISTRATIVE FILES

A. /etc/motd

This file contains the *message-of-the-day*. It is printed by */etc/profile* after every successful *login*; therefore, it should be kept short and to the point.

B. /etc/brc

This file is executed prior to entering any of the numbered *init* states for the first time after a reboot. The file is generally used to clear the file */etc/mnttab* and load KMC11s with their respective scripts. It is important to remember this file is executed once per reboot and is controlled by */etc/inittab*.

C. /etc/powerfail

This shell script is executed according to its line in */etc/inittab*. It is used primarily to reload

KMC11 scripts after a power failure has been detected.

D. /etc/rc

On the transition between init states, */etc/init* executes the shell script */etc/rc* (which must have executable modes). The execution of this file is controlled by a line in */etc/inittab*. For */etc/rc* to properly handle the mounting and unmounting of file systems, the opening of tty lines, etc., it may need certain information that is present in */etc/utmp*, namely, the new (current) state, the number of times this state has previously been entered, and the previous state. The following shell script fragment assigns this data to shell variables and checks for entering init state '2' for the first time. As an example:

```
set `who -r`
cur_mode=$7
no_times=$8
pre_mode=$9
if [ ${cur_mode} = 2 ) -a ${no_times} = 0 ) ]
then
# commands to be executed when entering multiuser mode
fi
```

Note that these values are carried over between reboots, that is to say, when the system is rebooted into a single user initial state, these values, stored in */etc/utmp* will reflect the state the system was in when it last went down.

Note that the files */etc/rc*, */etc/brc*, */etc/inittab*, */etc/powerfail*, and */etc/shutdown* must be edited to suit local conditions; see **brc**(1M).

E. /etc/inittab

This file is used by */etc/init* to determine which processes to create or terminate in each init state. By convention, state 's' is single user and state '2' is multiuser.

The following line may be used to indicate the default init state, that is, the state the system is to come up in (most likely single user):

```
is:s:initdefault:
```

The following lines arrange for appropriate execution of */etc/brc /etc/rc*, and */etc/powerfail*:

```
bc::bootwait:/etc/brc 1> /dev/console 2>&1
rc::wait:/etc/rc 1> /dev/console 2>&1
pf::powerfail:/etc/powerfail 1> /dev/console 2>&1
```

For line */dev/tty00* for use by 1200 baud asynchronous terminals, add the following:

```
00:2:respawn:/etc/getty -t60 tty00 1200
```

The arguments to getty are the number of seconds to allow before hanging up the line, the device name, optional speed settings which refers to an entry in */etc/gettydefs*, optional type of terminal referenced in **getty**(1M), and optional line discipline.

To add or delete *getty-login* processes while the system is in multiuser mode, make the appropriate changes to */etc/inittab* then issue the command **/etc/init q**. This forces */etc/init* to reread */etc/inittab* without having to change init states.

Again, this file must be edited for local conditions; see **getty**(8), **stgetty**(8), **init**(8), **gettydefs**(4), and **inittab**(5).

F. /etc/passwd

This file is used to describe each *user* to the system. A new line must be added for each new user. Each line has seven fields separated by colons:

1. Login name: normally 1 through 6 characters, first character alphabetic, the remainder alphanumeric, no uppercase characters.
2. Encrypted password: initially null, filled in by **passwd**(1). The encrypted password contains 13 bytes while the actual password is limited to a maximum of 8 bytes. The encrypted password may be followed by a comma and up to four more bytes of password "age" information.
3. User ID: a number between 0 and 65,535; 0 indicates the superuser. User IDs 0 through 99 are reserved.
4. Group ID: the default is group 1 (one). Group IDs 0 through 99 are reserved.
5. Accounting information: this field is used by various accounting programs. It usually contains the user name, department number, and account number.
6. Login directory: full pathname (keep them reasonably short).
7. Program name: if null, */bin/sh* is invoked after a successful login. If present, the named program is invoked in place of */bin/sh*.

For example:

```
ghh::138:1:6824-G.H.Hurtz(4357):/usr/ghh:
grk::244:1:6510-S.P.LeName(4466):/usr/grk:/bin/rsh
```

See also **passwd**(4), **login**(1), and **passwd**(1).

G. /etc/group

This file is used to describe each *group* to the system. The system administrator must add a new line for each new group. Each line has four fields separated by colons:

1. Group name: normally 1 through 6 characters, first character alphabetic, rest alphanumeric, no uppercase characters.
2. Encrypted password: contains 13 bytes while the actual password is limited to a maximum of 8 bytes.
3. Group ID: a number between 0 and 65,535. Group IDs 0 through 99 are reserved.
4. Login names: list of all *login* names in the group, separated by commas; list of all *login* names that may use **newgrp**(1) to become a member of the group.

Group passwords are strongly discouraged. See also **group**(4).

H. /etc/profile

When the shell is executed and is the leader of a process group, as is the case when it is invoked by *login*, it will read and execute the commands in */etc/profile* before executing commands in the user's *.profile* file. This allows the system administrator to set up a standard environment for all users (e.g., executing **umask**, setting shell variables, etc.) and take care of other housekeeping details (such as **news −n**). Note that in */etc/profile* the shell variable $0 indicates the invocation - normal shell (−sh), restricted shell (−rsh), or **su** command (−su).

I. /etc/checklist

This file contains a list of default devices to be checked for consistency by the **fsck**(1M) program. The devices normally correspond to those mounted when the system is in *multiuser* mode. For example, a sample checklist would be:

```
/dev/rp0
/dev/rrp1
```

Note that the *root* device is specified as a *block* device while all others are specified as *character* devices. Character devices can be checked faster than block devices. The *root* device is specified as a block device in order for the **fsck** program to detect when the *root* is being checked, so that any modifications to this file system will result in an immediate reboot request.

J. /etc/shutdown

This file contains procedures to gracefully shut down the system in preparation for file save or scheduled downtime. Beware that no procedures appear after the transition to single user mode as it may not be completed before the transition occurs.

K. /etc/filesave and /etc/tapesave

These files contain prototypes for local file saves.

L. /usr/adm/pacct

This file contains the process accounting information; see **acct**(1M).

M. /usr/lib/acct/holidays

This file defines the local company holiday schedule and designates the start of prime and non-prime processing hours for the accounting system.

N. /etc/wtmp

This file is the log of *login* processes.

REGENERATING SYSTEM SOFTWARE

System source is issued under the directory */usr/src*. The subdirectories are named *cmd* (commands), *lib* (libraries), *uts* (the operating system), *head* (header files), and *stand* (stand-alone programs); see **mk**(8) for details on how to remake system software.

NOTES

APPENDIX C

UNIX™ System V Init and Getty

UNIX™ System V Init and Getty

1. Introduction

In the UNIX* system environment, the initial process spawning is controlled and overseen by the first process forked by the UNIX operating system as it comes up at boot time. This process is known as *init*. One of the major jobs of *init* is to fork processes which will become the *getty-login-sh* sequence. This sequence of processes allows users to *login* and takes care of setting up the initial conditions on the outgoing terminal lines so that the speed and the other terminal related states are correct. *Init* and these other processes also keep an accounting file **/etc/wtmp** that is available to processes on the system. With these files it is possible to determine the state of each process that *init* has spawned, and if it is a terminal line, who the current user is. One program in particular, *who*(1), provides a means of examining these files.

This document describes the capabilities of each program used in this new implementation, the databases involved, and how to create and maintain these databases. In addition, the debugging features designed into both *init* and *getty* are described in the event remedial action is required or modifications are attempted.

2. Init

Init is driven by a database, its previous internal level, its current internal level, and events which cause it to wake up.

2.1 The Database: /etc/inittab

Init's database, kept in the file **/etc/inittab**, consists of any number of separate entries, each with the form:

id:level:type:process

id — The *id* is a one to four letter identifier which is used by *init* internally to label entries in its process table. It is also placed in the dynamic record file, **/etc/utmp**, and the history file, **/etc/wtmp**. The *id* should be unique.

level — The *level* specifies at which levels *init* should be concerned with this entry. *Level* is a string of characters consisting of [0-6a-c]. Anytime that *init*'s internal level matches a level specified by *level*, this entry is *active*. If *init*'s internal level does not match any of the levels specified, then *init* makes certain that the process is not running. If the level field is empty it is equivalent to the string "0123456".

type — The *type* specifies some further condition required for or by the execution of an entry.

- off — The entry is not to run even if the levels match.
- once — The entry is to be run only if *init* is entering a level. This means if *init* has been awakened by powerfail or because a child died this entry will not be activated. Only when a user signal requests a change of *init*'s internal state to a state which is different from its current state, and this new state is one in which this entry should be active, will this entry be activated.

---

* UNIX is a Trademark of Bell Telephone Laboratories, Incorporated.

wait — *Wait* has all the characteristics of *once*, plus it causes *init* to wait until the process spawned dies before reading anymore entries from its database. This allows for initialization actions to be performed and completed before allowing other processes which might be affected to start running. It is common in the OSS environment for shared memory segments to be initialized this way and semaphores to be conditioned.

respawn — *Respawn* requests that this entry continue to run as long as *init* is running in a level which is in this entry's *level* field. Most processes spawned by *init* fall into this category. All *getty* processes are marked as *respawn*. Whenever *init* detects the death of a process that was marked *respawn*, it spawns a new process to take its place.

boot — *Boot* entries have the execution behavior of *once* entries. They are started only when *init* is switching to a numeric run state for the first time. Most commonly *boot* entries have an empty *level* string, meaning that no matter which level *init* switches to the first time, the *boot* entry will be run. Should there be a more specific *level* string, for example "01", then the *boot* entry would only be run if *init* switched to either the 0 or 1 run state as its first numeric level.

bootwait — *Bootwait* entries have the execution behavior of *wait* entries and they, like *boot* entries, are only run as *init* switches to a numeric level for the first time.

power — *Power* entries act like *once* entries and are activated if *init* receives a SIGPWR signal (19) and is in a state which matches the active states for the entry.

powerwait — *Powerwait* entries act like *wait* entries and are activated if *init* receives a SIGPWR signal and is in a state which matches the active states for the entry.

initdefault — *Initdefault* is a non-standard entry in that it does not specify some process to be spawned. Instead it only specifies which level *init* is to go to initially when it is coming up at boot time. This allows the system to be rebooted without an operator having to make entries at the system console if so desired. If there is no *initdefault* entry, then *init* will ask at the system console, **/dev/syscon**, for the initial run state. In addition to specifying the numbered states, the single-user state [s] may also be specified.

process — The *process* field is the action that *init* will ask a *sh* to perform whenever the entry is activated. The string in the **process** field is given a prefix of "exec " so that each entry will only generate one process initially. *Init* then forks and execs

```
sh -c "exec process"
```

This means that the **process** string can take full advantage of all **sh** syntax. The only peculiarities arise from the string "exec ", which was prefixed to the string, and because initially there is no standard input, output, or error output. The addition of "exec " to the string means that if the user wants to have a single entry generate more than one process, for example making a list of the people on the system at the time of a powerfail and mailing it to **root** by the command "who | mail root", it would have to be put in as

```
pf::powerwait:sh -c "who | mail root"
```

to work. If it was put in simply as "who | mail root", it would be executed as "exec who | mail root", and only the *who* process would be created before the *sh* disappeared. The

lack of standard input and output channels must be addressed by explicitly specifying them. An example is the **blog** program that many OSS's run as a *bootwait* entry as the system comes up. Since it requires the operator to supply input, it appears as

bl::bootwait:/etc/blog </dev/syscon >/dev/syscon 2>&1

in **/etc/inittab**.

2.2 Levels

A level is one of seven numeric levels, denoted 0, 1, 2, 3, 4, 5, or 6, three temporary levels, denoted a, b, or c, or the single-user level, s. Normally *init* runs in a numeric level. Precisely how a particular level is used depends entirely on the database and the system administrator. The temporary levels allow certain entries to be started on demand without affecting any processes that were started at a particular level. The temporary levels immediately revert to the previous numeric level once all entries in the database have been scanned to see if they should be started at the temporary level. When an entry is started by a switch to a temporary level, it becomes independent of future level changes by *init*, except a change to the single-user level. The only way to kill a process that was started as a respawnable demand process, without going to the single-user level, is to modify the database, declaring the entry to be *off*.

The single-user level is the one level independent of the database. For this reason it is not a level in the normal sense. In the single-user level *init* spawns off a *su* process on the system console, and that is the only process that it maintains while at the single-user level. The single-user level can be entered at two different places in *init*. If it is entered at boot time it allows the operator to look over the file systems without having *init* attempt to do any file I/O, which might cause further problems. *Init* will not attempt to recreate **/etc/utmp** or access **/etc/wtmp** until after it has left this initial single-user level. If the single-user level is entered at any other time, *init* does do the bookkeepping in the record files.

The system administrator requests *init* to change levels by running a secondary copy of *init* itself. **/etc/init** is linked to **/bin/telinit**, and it is usually through the *telinit* name that this is accomplished. *Init* can only be run by root or a privileged group. Whenever *init* starts running and finds that its process id is not 1, it assumes that it is a user initiated copy, which is supposed to send a signal to the real *init*. The usage is:

telinit [0123456sSqQabc]

and the single character argument specifies the signal to be sent to *init*. If the request is to switch to the single-user level, 'S' or 's', then *init* also relinks **/dev/syscon** to the terminal originating the request so that it becomes the virtual system console, thus insuring that future messages from *init* will be directed to the terminal where the operator is located. When it does this relinking it also sends a message to **/dev/systty**, saying that the console is being relinked to some other terminal so that there is a record of the fact at the physical system console.

2.3 Waking Events

There are four events which will wake *init*: boot, a powerfail, death of a child process, or a user signal.

boot *Init* operates in the **boot** state until it has entered a numeric state for the first time. It is not possible for *init* to reenter the boot state a second time. Commands labeled *boot* and *bootwait* are executed when changing to a numeric state for the first time, if the levels match.

powerfail Any time power fails, the operating system sends a SIGPWR signal to all processes. *Init* will execute commands with types of *power* and *powerfail*.

child death Any time a child process of *init* dies, *init* receives a SIGCLD signal (18). The dead child process may be one of two types, a direct decendent of *init*, or a process whose own parent process died before it did. The parent of a process automatically

becomes *init*, if its real parent should die before it does. *Init* determines immediately if the defunct process was one of its own children or an *orphan*. If it was one of its own, it performs the necessary bookkeepping on its internal process table to note that the process died. If *init* was busy at the time it received the SIGCLD signal, it then returns to complete whatever action it was performing. If *init* was asleep, it then scans its database to determine if any other actions should be taken, such as respawning the process.

user signal *Init* catches all signals that it is possible for a process to catch. Most signals have specific meaning to *init*, usually requesting it to change its current state in some way. There is one signal, the 'Q' signal, which is used just to waken *init* and cause it to scan its database. This is often issued after a change has been made to the database so that *init* will put the new change into effect immediately. If this was not done, the change would not become effective until *init* had wakened for some other reason. Other than during the initialization phase, it is solely with signals that the system administrator controls the internal level at which *init* is running.

2.4 Normal Operational Behavior

Init scans **/etc/inittab** once or twice for each event which wakes it up. If it is in the *boot* or *powerfail* state, it scans the table once, looking for entries of these types, and then switches itself back to a *normal* state and scans again.

Its first action in the normal state is to scan **/etc/inittab** and remove all processes which are currently active and should not be at the current level. *Init* employs one of two methods when killing its child processes depending on whether it is changing levels or not. If *init* is not changing levels, it forks a child process for each child that needs to be killed, and has that child process send the signals to the process targeted for extinction. Killing a process involves sending it two signals. First a SIGTERM signal (15), is sent so that it can clean up after itself and die gracefully. After waiting the amount of time defined as TWARN (the default value is 20 seconds), a SIGKILL signal (9), is sent, which guarantees that the child will die, if it hasn't done so already. Forking a child to do the killing has the advantage that the main *init* process need not wait for all the processes it is killing to die before beginning the spawning of new processes. The disadvantage is that if many processes were being killed this way, there would be a very real chance of the operating system process table filling up, which causes the *fork* system call to fail. This in turn would upset *init* at the very least and cause it to have to wait anyway. For this reason, when *init* is changing levels, it assumes that it may have many processes to terminate and so it sends the signals itself, waits for the required 20 seconds, and sends the final termination signals, before continuing. Once the old processes have been removed, *init* makes an entry in its accounting files if it is changing levels. At this point it either enters the single-user level or rescans its database looking for processes that need to be spawned at the current level and in the current state. In the normal state of operation *init* is looking for entries whose types are *off*, *once*, *wait*, or *respawn*.

With the completion of the scan of the database in the normal state, *init* is ready to wait for another event. To ensure that a user who just logged off has had his or her files updated to the disk and to insure that the bookkeepping is also updated to the disk, *init* performs a *sync* system call and then pauses until it is awakened again for some new reason.

If *init* finds that it is being requested to switch to the single-user level when it wakens from the pause, it saves all the *ioctl* information about the system console in the file */etc/ioctl.syscon* before proceeding to remove all its other children. It does this so that if the system is being taken down, the new *init* process will know how to set up the system console to talk to it. It is a convenient feature to not have to change the baud rate and terminal specifications if you are rebooting a system remotely. Because *init* preserves the ioctl state of the system console across system reboots, messages coming out during reboots are legible to the operator, no matter where the system console happens to be linked.

All written messages from *init* are sent to */dev/syscon*. In reality, *init* itself does not send the message, but forks a child to send the message. This is because *init* must never open a terminal line or it will be assigned a controlling terminal. Since *init* has no controlling terminal, it can spawn *getty* processes which initially have no controlling terminal. When such a *getty* opens its assigned terminal, the terminal becomes the controlling terminal for it and its children. In the one instance *init* needs input from the system administrator during the initialization phase. In this case, the child process which is asking for the run level opens */dev/systty*, which is always the physical system console, before opening */dev/syscon*, the virtual system console. This causes */dev/systty* to be the child's controlling terminal. Thus, should the computer be coming up, */dev/syscon* not be linked to */dev/systty*, and */dev/syscon* be down (perhaps because the datalink went down during the reboot), it is possible for a person at */dev/systty* to regain control by typing a <DEL> character. This causes a SIGINT signal (2) to be sent to the child process, which will relink */dev/systty* to */dev/syscon* and ask again for a run level, this time at the physical system console.

2.5 Setting Tunable Variables

Init has several tunable timing constants that can be adjusted when it is compiled.

SLEEPTIME *Init* guarantees that it will awaken occasionally even if the system is quite inactive. It does this by setting an alarm timer before going to sleep. The length of that timer is defined by SLEEPTIME, and is initially five minutes. Since *init* does a *sync* system call each time it wakes, this guarantees that there will be a *sync* at least once every SLEEPTIME seconds.

TWARN TWARN is the number of seconds between the SIGTERM signal and the SIGKILL signal, when *init* is removing processes. It should be set long enough so that all processes who want to, can die gracefully on receipt of the SIGTERM signal. It is initially 20 seconds.

NPROC This is the size of the internal process table *init* uses to keep track of its child processes. It currently defaults to 100, though it can be passed in during compilation with the -D option. I recommend you set it to the size of the system's process table.

WARNFREQUENCY To prevent *init* from flooding the system console with error messages when it own internal process table is full, *init* only generates an error message once each WARNFREQUENCY times that it is unable to find a slot. Proper sizing of the internal process table should prevent this condition from ever occurring.

Init cannot directly tell if there is something wrong when it tries to fork and exec a command. It assumes that there is something wrong if it has to respawn a particular entry too often. There are three related defines controlling this feature, SPAWN_LIMIT, SPAWN_INTERVAL, and INHIBIT.

SPAWN_LIMIT SPAWN_LIMIT is the number of times a process may respawn in a certain interval of time before further respawns are inhibited.

SPAWN_INTERVAL SPAWN_INTERVAL is the interval of time in seconds that SPAWN_LIMIT number of respawns must occur to cause inhibition of an entry. If an entry should respawn too often, a message is generated on the system console indicating which line in **/etc/inittab** is at fault.

INHIBIT INHIBIT is the number of seconds of inhibition that will be applied to a process which has respawned too often.

SPAWN_LIMIT and SPAWN_INTERVAL should be set so that it is possible for *init* to respawn a process fast enough to cause inhibition, but not so low that it is possible to have a legal death of a process happen so rapidly that it is inhibited. The current limits are ten respawns in two minutes. The real problem is that when something like *getty* disappears, *init* becomes active trying to respawn many processes and never gets to respawn a single process often enough to set off the alarm. The INHIBIT limit is five minutes. Once an entry is inhibited, it is possible to restart it sooner than

INHIBIT seconds later by sending *init* the 'Q' signal. The normal problem is a typo in **/etc/inittab**, and the normal procedure is to correct the typo and then do a "telinit Q" to cause *init* to attempt the spawning entry again.

2.6 Debugging Features

Init has some debugging features built in. There are three conditional debug flags, which allow various flavors of debugging to be enabled.

UDEBUG This flag causes *init* to be compiled in a form that can be run as a normal user process instead of as process 1. This allows a person to use *sdb* on it in a normal fashion and to not disturb the rest of the system while debugging or modifications are made and tested. There are differences in this user version of *init*. It assumes that **utmp**, **wtmp**, **inittab**, **ioctl.syscon**, and **debug** are all in the local directory instead of **/etc**. It also writes to **/dev/sysconx** and **/dev/systtyx**, instead of **/dev/syscon** and **/dev/systty**. It does not process all signals in the same fashion that the real *init* does. Signals SIGINT, SIGQUIT, SIGIOT, and SIGTERM, which correspond to the signals to change to levels 2, 3, 4, and ignore are left in their default modes, so that it is possible to terminate the user "init" from a terminal. Signals SIGUSR1 and SIGUSR2, which are normally ignored by the real *init* are set to cause an *abort* for capturing cores of the debug *init*. The UDEBUG flag automatically sets the DEBUG flag, meaning that the first level of debug will be generated by the *init* and written into the file *debug* in the current directory.

DEBUG This flag causes a version of *init* to be produced that can be run as the real *init*, but which generates diagnostic messages about process removal, level changes, and accounting and writes them in the file **/etc/debug**.

DEBUG1 DEBUG1 causes the diagnostic output generated by DEBUG1 to be increased substantially. Specifically it produces messages about each process being spawned from *inittab*.

3. Getty

Getty is responsible for making appropriate setting of terminal characteristics and baud rate so that a user can communicate with the UNIX system. The most important of those features is the choice of a baud rate so that input and output make sense. In the old version of *getty*, there was a hardwired table in *getty* which controlled the search for the correct speed. The starting point in the search is specified by the arguments passed to *getty*. If there was some reason to change the baud rate search, *getty* had to be modified itself, and recompiled. In the new *getty*, the search is controlled by an ascii file, **/etc/gettydefs**, and changing or augmenting the search behavior only requires that the file be edited.

3.1 Usage

Getty is normally started from **/etc/inittab** by *init*. *Getty* takes from one to six arguments:

getty [-h] [-t time] line [speed_label] [term_type] [line_disc]

−h This switch tells *getty* that it should not drop the Data Terminal Ready signal before resetting the line. This switch currently only works in the CB-UNIX system environment. Normally *getty* ensures that DTR goes down so that connections to the Develcon dataswitch will be disconnected everytime. The EIA protocol requires that a dataset see DTR drop and be reasserted before answering another call. It is possible for *getty* to come back on a line before all the processes spun off by the previous user have died and closed their connections to the line. In this case, DTR would not drop if *getty* didn't insure it. This switch is required for programs like *ct*, which initiate a call from the computer to a user (instead of the user calling the computer), putting a *getty* on the resulting connected line. Without the -h switch, the *getty* would immediately disconnect the user again.

−t This switch specifies that the *getty* should die after the specified number of seconds if nothing is typed. This prevents datasets from being tied up if someone isn't actually logging in after they've gotten connected.

line *Line* is the name of the terminal line, which *getty* is to open and set up. It is minus **/dev/** since *getty* does a *chdir* to the **/dev** directory and expects to find it in that directory.

speed_label The *speed_label* is usually something like "1200" or "9600", which appears to directly specify a baud rate, but in reality can be anything since it really is a label of an entry in **/etc/gettydefs** for which *getty* looks. It specifies the entry *getty* will start with when trying to find an appropriate speed to for the terminal. It defaults to "300" if there is none given.

term_type The *term_type* specifies which terminal discipline is to be used. If this is specified, the virtual terminal protocol becomes immediately effective on the line. Typical types might be "vt100", "hp45", or "tek". Whatever type is specified, it must be a terminal handler that has been compiled into the operating system to be effective. This argument is given for lines that are hardwired to the computer.

line_disc The *line discipline* is the last thing that can be specified. The most common is "half" or "half_duplex", when there is a half duplex terminal coming into the computer. This causes the appropriate line discipline to be associated with the line.

3.2 The Database: /etc/gettydefs

Whenever *getty* is invoked it references its database to determine certain information about how to set up the line. Each entry in the database has a fixed format.

label# initial flags # final flags # login msg #nextlabel

Getty matches its *speed_label* argument against the "label" field. It stops searching when it finds an entry with a label that matches. The entry specifies how the terminal is supposed to be setup during the initial phase, the phase when *getty* prints out the "login msg" and reads in the user's login name, and the final phase, when *getty* exec's the *login* program to continue to the *login* process. The baud rate is specified as an *ioctl* flag in both the initial and final flags fields.

The flags themselves are strings matching the define variables found in **/usr/include/termio.h**. It should be noted that these flags may be partially or totally overridden if there is a terminal type specified. When a terminal type is enabled, it resets various flags to suitable conditions automatically.

During the initial phase, *getty* always puts the terminal into a non-echoing *raw* mode. This allows it to take each character as it comes in and infer certain things about the terminal. For instance, if it sees upper case alphabetic characters, but no lower case, it then assumes that the terminal is upper case only and sets it up in the final configuration so that the upper to lower case conversions are made. Also if the speed is wrong it will get a <NULL> character (or <ESC><NULL> character if a terminal type is set) if there is a framing or parity error. This means that the speed is wrong and another speed should be tried.

The typical "initial flags" would only include the speed, for example "B1200 CS7 PARENB HUPCL". "CS7 PARENB" sets the line for 7 bits, even parity characters. "HUPCL sets the line to hangup on close. Typical final flags would be "B1200 SANE IXANY TAB3". "SANE" is not a real flag found in the header file, but a collection of *ioctl* flags used for normal terminal behavior. "IXANY" permits the use of any character to restart output. "TAB3" says to expand tabs on output.

The "login msg" field is the message that *getty* will print before waiting for the user to enter his or her login name. It may contain anything desired and *getty* understands normal special character conventions so that "\n" means <lf> as does "\012". On systems that are not using the terminal

handlers and where lines are hardwired, people have been known to make up special entries for different terminal types, for example:

```
vt100-2400# B2400 # B2400 SANE TAB3 7953OGIN: #vt100-1200
        # 33[H 33[2JAMACCS System B
```

where the "login msg" contains the special vt100 characters required to clear the screen. Notice also that the entry can take more than one line. Entries are delimited by a blank line. Lines that begin with a pound sign (#) are ignored so that comments may be added to the file.

The "next_label" field tells *getty* which entry to try next if it gets an indication that the speed is wrong. In the above example it would look for an entry with the name "vt100-1200" if this one wasn't at the proper speed. Normally the entries don't contain terminal specific information, and the various speed choices are linked together in a closed circle of some sort. For example it is common to have 9600 -> 4800 -> 2400 -> 1200 -> 300 -> 9600. In this way, no matter where you enter the circle, sooner or later you should be able to get to the speed that is correct for your terminal.

To enable the system administrator to check the database for readability by *getty*, there is a checking mode in which *getty* can be run.

```
getty -c gettydefs_like_file
```

When *getty* is run in this mode, it scans the entire input file specified and deciphers each entry, printing out the resulting modes that it will set. If it finds a line that it cannot read, it prints an appropriate message, which allows the administrator to correct the entry. By this mechanism it is possible to avoid installing a misformatted gettydefs file and have it tie up the system.

Also as a safety measure, should *getty* be unable to find **/etc/gettydefs**, it does have a one fallback entry built in. Should *gettydefs* disappear for some reason, a user could still log in at 300 baud, since this is the default setting in the built-in entry.

3.3 Operational Behavior

As has been shown earlier, *getty* sets up a line as specified by an entry from **/etc/gettydefs** and from any additional arguments, outputs the "login msg" field, and then tries to read the user's login name from the line. During the input of the login name, getty checks for speed mismatches that the operating system will report as a <NULL> character. If such a mismatch occurs, *getty* tries the next speed specified by the current entry, and repeats the whole sequence. Also while reading in the login name, *getty* makes a guess whether the terminal is upper case only. If it sees some upper case characters, but no lower case characters, it assumes that the terminal is upper case only and sets the *ioctl* state of the line to translate upper case letters to lower case on input, and lower to upper case on output.

An addition has been made to *getty* and *login*, which allows for environmental variables to be set up at the time a user enters his or her login name. This allows users to control the behavior of their *.profile* at the time they specify their login names. *Getty* executes the *login* program by passing all the separate words given it in response to the login message as arguments to *login*. If for example, the user responded with "jls f", then *getty* would execute "login jls f" as its final action. See the *login* section to see how this modifies the commands behavior.

4. login

Unlike *init* and *getty*, *login* did not require a great deal of modification. The only required change was that it should write to **/etc/utmp** and **/etc/wtmp** in the new format. This change was minor. At the time this change was made, a change visible to the user was also made: the ability to add to the environment. This change was added as a convenience. It allows the user to modify the behavior of his or her *.profile* by having environmental variables set which the *.profile* script knows about.

The basic change was that any additional words provided in response to the basic "login:" query are placed in the environment of the *sh* executed by *login* as its last act in the following way. If the word does not contain an '=', a shell variable of the type "Ln=word" is created. "n" is a number starting at 0 and for each new environment variable it is incremented by one. If the word does contain an '=', then the whole string is passed in the environment unchanged. For example, "TERM=2621" would be placed in the environment unchanged and the shell variable $TERM would be defined as "2621".

To preserve security, there are a couple of exceptions. It is not possible to change the shell variables $PATH or $SHELL by this mechanism. That means that a restricted shell will remain restricted and that the user cannot gain access to commands that might allow him to avoid the usual restrictions of *rsh*.

5. who

Who(1) is the program that reads the history files maintained by *init*, *getty*, and *login*. Since the format of these files was changed substantially, it was necessary to change *who*. In the process some additional features where added to *who* so that it would convey more useful information to users. The standard usage for *who* is:

who [-uTlpdbrtas] [[am i] or [utmp_like_file]]

u This returns a listing of useful information for all the users. This information includes login time, activity, pid and comment from inittab file.

T Report the writability state of the terminal for that entry.

l Report all entries that are living *getty* processes.

p Report all entries for living children of *init* excluding *getty* and decendents of *getty*.

d Report all the entries for processes that have died.

b Report the boot time entries that *init* has made. In **/etc/utmp** there is only one such entry.

r Report the run level entries that *init* has made. In **/etc/utmp** there is only one such entry, the current run level entry. The current state, the number of times in that state, and the previous state are also reported.

t Report the change of date entries that have been made by the *date*(1) command when the clock was reset. These are required in the history file, **/etc/wtmp**, if accounting is to be done.

a Report all the entries.

s Report information for all users in short form, this is the default.

If no file is specified, then **/etc/utmp** is assumed. The **who am i** sequence returns the entry for the user typing the command.

There are various output formats for the different kinds of entry. In particular, entries for users and *getty* processes list the amount of time since output to the terminal occurred. This is often of interest since it shows other users whether someone is actually working at a terminal or not. The comment field at the end of the entry from **/etc/inittab** is also included, which can conveniently be set up to be the location of the terminal. Dead entries report the exit status for the process that died. This can be of use, since it shows whether the process terminated abnormally or not.

6. Other Affected Programs

All programs accessing the accounting files were affected by the new utmp structure. In particular, *date*(1) makes two entries indicating the old time and new time, whenever it changes the system clock. Also affected are the commands in **/usr/lib/acct**, which produces reports based on the information in **/etc/wtmp.**

7. utmp format

A major change in going to the new *init* was that it uses a different format in writing out its records in **/etc/utmp** and **/etc/wtmp.** The new format is:

```
/*      <sys/types.h> must be included.                                  */

#define UTMP_FILE     "/etc/utmp"
#define WTMP_FILE     "/etc/wtmp"
#define ut_name       ut_user

struct utmp
 {
        char ut_user[8] ;          /* User login name */
        char ut_id[4] ;            /* /etc/lines id(usually line #) */
        char ut_line[12] ;                 /* device name (console, lnxx) */
        short ut_pid ;                     /* process id */
        short ut_type ;            /* type of entry */
        struct exit_status
         {
          short e_termination ;    /* Process termination status */
          short e_exit ;           /* Process exit status */
         }
        ut_exit ;                  /* The exit status of a process
                                           * marked as DEAD_PROCESS.
                                           */
        time_t ut_time ;           /* time entry was made */
 } ;

/*      Definitions for ut_type                                          */

#define EMPTY         0
#define RUN_LVL               1
#define BOOT_TIME     2
#define OLD_TIME      3
#define NEW_TIME      4
#define INIT_PROCESS  5            /* Process spawned by "init" */
#define LOGIN_PROCESS         6          /* A "getty" process waiting for login */
#define USER_PROCESS7          /* A user process */
#define DEAD_PROCESS          8
#define ACCOUNTING  9

#define UTMAXTYPE   ACCOUNTING  /* Largest legal value of ut_type */

/*      Special strings or formats used in the "ut_line" field when  */
/*      accounting for something other than a process.               */
/*      No string for the ut_line field can be more than 11 chars +  */
/*      a NULL in length.                                            */

#define RUNLVL_MSG  "run-level %c"
#define BOOT_MSG    "system boot"
#define OTIME_MSG   "old time"
#define NTIME_MSG   "new time"
```

The *ut_type* field completely identifies the type of entry, the *ut_id* field only contains the "id" as found in the "id" field of **/etc/inittab.** The *ut_line* field was expanded and freed so that it can

contain things like *console* or other things that are not of the form **/dev/ln***xx*. Finally *ut_exit* contains the exit status of processes that *init* has spawned and that have subsequently died.

APPENDIX D

An Introduction to Display Editing with Vi

An Introduction to Display Editing with Vi©

The **vi** (visual) editor† is a display-oriented, interactive text editor. When using **vi**, the screen of your terminal acts as a window for looking into the file which you are editing. Changes which you make to the file are reflected in what you see. The **ex** editor supports both command and display-oriented editing. It is line-oriented.

Using **vi** you can insert new text into the file at any place quite easily. Most of the commands to **vi** move the cursor around in the file. There are commands to move the cursor forward and backward in units of characters, words, sentences, and paragraphs. A small set of operators, like **d** for delete and **c** for change, are combined with the motion commands to form operations, such as "delete a word" or "change a paragraph", in a simple and natural way. This regularity and the mnemonic assignment of commands to keys makes the editor command set easy to remember and use.

The **vi** editor will work on a large number of display terminals, and new terminals are easily driven after editing a terminal description file. While it is advantageous to have an intelligent terminal which can locally insert and delete lines and characters from the display, the editor functions quite well on dumb terminals over slow telephone lines. The editor makes allowance for the low bandwidth in these situations and uses smaller window sizes and different display updating algorithms to make best use of the limited speed available.

It is also possible to use the command set of the **vi** editor on hard copy terminals, storage tubes, and "glass tty's" using a one line editing window; thus the **vi** command set is available on all terminals. The full command set of the more traditional line-oriented editor **ex** is available within **vi**. It is quite simple to switch between the two modes of editing.

© Copyright 1979, 1980, 1982 Regents of the University of California, reprinted by permission.

† The *vi(ex)* editor is based on software developed by The University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

1. Introduction

This document provides an introduction to the **vi** (visual) editor, versions 2 and 3.[1] You should be running **vi** on a file you are familiar with while reading this. Sections 2 through 6 describe the basics for using **vi** and include the display editing features of the **ex** editor. Some topics of special interest are presented in sections 7 and 8, and some details of how the editor functions are saved for section 9 to avoid cluttering the presentation here. A reference manual describing the command-oriented part of the **ex** editor is presented in sections 10 through 18. Sections 10 through 15 contain command, file, and editing mode information for the **ex** editor. Command descriptions, regular expressions, and option descriptions are provided in sections 16, 17, and 18.

At the end of sections 3 and 4 a summary of commands, control characters, and key functions is discussed. The summary gives the name of command, paragraph of reference, and a short description. Table A provides a complete list of characters and their special meanings to the **vi** editor. There is also a quick reference card available which summarizes the commands of **vi** in a very compact format.

In the command examples shown in this part:

- Input which must be typed "as is" will be presented in **bold face**.
- Text which should be replaced with appropriate input will be given in *italics*.

2. Getting Started

2.1 Specifying Terminal Type

Before you can start **vi** you must tell the system what kind of terminal you are using. Here is an incomplete list of terminal type codes. If your terminal does not appear here, consult with one of the staff members on your system to determine the code for your terminal. If your terminal does not have a code, one can be assigned and a description for the terminal created.

CODE	FULL NAME	TYPE
2621	Hewlett-Packard 2621A/P	Intelligent
2645	Hewlett-Packard 264x	Intelligent
act4	Microterm ACT-IV	Dumb
act5	Microterm ACT-V	Dumb
adm3a	Lear Siegler ADM-3A	Dumb
adm31	Lear Siegler ADM-31	Intelligent
c100	Human Design Concept 100	Intelligent
dm1520	Datamedia 1520	Dumb
dm2500	Datamedia 2500	Intelligent
dm3025	Datamedia 3025	Intelligent
fox	Perkin-Elmer Fox	Dumb
h1500	Hazeltine 1500	Intelligent
h19	Heathkit h19	Intelligent
i100	Infoton 100	Intelligent
mime	Imitating a smart act4	Intelligent
t1061	Teleray 1061	Intelligent
vt52	Dec VT-52	Dumb

1. Version 2 is the version of **vi** which runs on the PDP11, hereafter referred to as the V2 editor. Version 3 runs on 32-bit machines, hereafter referred to as the V3 editor.

Suppose for example that you have a Hewlett-Packard HP2621A terminal. The code used by the system for this terminal is '2621'. The command sequence

TERM=2621
export TERM

would tell the system you have a Hewlett-Packard 2621A/P.

If you want to have your terminal type established automatically when you log in, place the above commands in your *.profile*.

2.2 Editing a File

After telling the system which kind of terminal you have, make a copy of a file you are familiar with and run **vi** on this file with the command

$ vi *filename*

Replace *filename* with the name of the copy file just created. The screen should clear and the text of your file appear on the screen. If something else happens perhaps you gave the system an incorrect terminal type code, in which case the editor may have just made a mess out of your screen. This happens when **vi** sends control codes for one kind of terminal to some other kind of terminal. In this case hit the keys **:q** (colon and the q key) and then the RETURN key. This should get you back to the command level interpreter. Figure out what you did wrong (ask someone else if necessary) and try again.

Another thing which can go wrong is that you typed the wrong file name and the editor just printed an error diagnostic. In this case you should follow the above procedure for getting out of the editor; then try again, spelling the file name correctly.

If the editor doesn't seem to respond to the commands which you type here, try sending an interrupt to it by hitting the DEL or RUB key on your terminal, and then hitting the **:q** command, again followed by a carriage return.

2.3 Editor Copy in Buffer

The **vi** editor does not directly modify the file which you are editing. Rather, it makes a copy of this file in a place called the *buffer* and remembers the file's name. You do not affect the contents of the file unless and until you write the changes made back into the original file.

2.4 Arrow Keys

The editor command set is independent of the terminal you are using. On most terminals with cursor positioning keys, these keys will also work within the editor. If you don't have cursor positioning keys, or even if you do, you can use the **h j k** and **l** keys as cursor positioning keys (these are labelled with arrows on an *adm3a*).

- *h* moves cursor to the left (like control-h which is a backspace)
- *j* moves cursor down (in the same column)
- *k* moves cursor up (in the same column)
- *l* moves cursor to the right.

Note: On the HP2621 terminal the function keys must be shifted to send to the machine, otherwise they only act locally. Unshifted use will leave the cursor positioned incorrectly.

2.5 Special Characters

Several special characters are very important, so be sure to find them right now. Look on your keyboard for a key labelled ESC or ALT. It should be near the upper left corner of your terminal. Try hitting this key a few times. The editor will ring the bell to indicate that it is in a quiescent state. On smart terminals where it is possible, the editor will quietly flash the screen rather than

ring the bell. Partially formed commands are cancelled with the ESC key. When you insert text in the file, text insertion is ended with the ESC key. This key is a fairly harmless one to hit, so you can just hit it if you don't know what is going on.

The CR or RETURN key is important because it is used to terminate certain commands. It is usually at the right side of the keyboard and is the same command used at the end of each shell command.

Another useful key is the DEL (or RUB key). It generates an interrupt, which tells the editor to stop what it is doing. This is a forceful way of making the editor listen to you, or to return to the quiescent state if you don't know or don't like what is going on. Try hitting the '/' key on your terminal. It is used when you want to specify a search string. The cursor should now be positioned at the bottom line of the terminal after a '/' printed as a prompt. You can get the cursor back to the current position by hitting the DEL or RUB key. Backspacing over the '/' will also cancel the search. From now on we will simply refer to hitting the DEL or RUB key as "sending an interrupt."[2]

The editor often echoes your commands on the last line of the terminal. If the cursor is on the first position of this last line, then the editor is performing a computation, such as computing a new position in the file after a search or running a command to reformat part of the buffer. When this is happening you can stop the editor by sending an interrupt.

2.6 Leaving the Editor

After you have worked with this introduction for a while, and you wish to do something else, you can give the ZZ command to leave the editor. This writes the contents of the editor buffer (including any changes made) back into the file you are editing and then quits the editor. You can also end an editor session by giving the command **:q!**CR.[3] This is a dangerous but occasionally essential command which ends the editor session and discards all your changes. You need to know about this command in case you change the editor copy of a file you wish only to look at. Be very careful **not** to give this command when you really want to save the changes you have made.

3. Moving Around in the File

3.1 Scrolling and Paging

The editor has a number of commands for moving around in the file. The most useful of these is generated by hitting the control and D keys at the same time, a control-D or '^D'. We will use this two-character notation for referring to these control keys from now on. You may have a key labelled '^' on your terminal. This key will be represented as '↑' in this document; the '^' is exclusively used as part of the '^x' notation for control characters. If you don't have a '^' key on your terminal then there is probably a key labelled '↑'; these characters are one and the same.

As you know now, if you tried hitting **^D**, this command scrolls down in the file. The **D** thus stands for down. Many editor commands are mnemonic and this makes them much easier to remember. For instance the command to scroll up is **^U**. Many dumb terminals can't scroll up at all, in which case hitting **^U** clears the screen and refreshes it with a line which is farther back in the file at the top.

If you want to see more of the file below where you are, you can hit **^E** to expose one more line at the bottom of the screen, leaving the cursor where it is.[4] The command **^Y** (which is hopelessly non-mnemonic, but next to **^U** on the keyboard) exposes one more line at the top of the screen.

---

2. On some systems, this interrupt feature comes at a price; you cannot type ahead when the editor is computing with the cursor on the bottom line.

3. All commands which read from the last display line can also be terminated with an ESC as well as a CR character.

4. Version 3 only.

There are other ways to move around in the file; the keys **^F** and **^B** move forward and backward a page, keeping a couple of lines of continuity between screens so that it is possible to read through a file using these rather than the **^D** and **^U** if you wish.

Notice the difference between scrolling and paging. If you are trying to read the text in a file, hitting **^F** to move forward a page gives you only a little context at which to look back. Scrolling on the other hand gives more context and functions more smoothly. You can continue to read the text while scrolling is taking place.

3.2 Searching, goto, and Previous Context

Another way to position yourself in the file is by giving the editor a string to search for. Type the character / followed by a string of characters terminated by a RETURN. The editor will position the cursor at the next occurrence of this string. Then try hitting **n** to go on to the next occurrence of this string. The character **?** will search backward from where you are and is otherwise like /.

These searches will normally wrap around the end of the file, and thus find the string even if it is not on a line in the direction you search provided it is anywhere else in the file. You can disable this wrap-around in scans by giving the command:

:se nowrapscanCR
or
:se nowsCR

If the search string you give the editor is not present in the file, the editor will print a diagnostic on the last line of the screen and the cursor will return to its initial position.

If you wish the search to match only at the beginning of a line, begin the search string with an ↑. To match only at the end of a line, end the search string with a **$**. Thus

/↑firstCR

will search for the word 'first' at the beginning of a line, and

/last$CR

searches for the word 'last' at the end of a line.

Actually, the string you give to search for can be a *regular expression* in the sense of the editors **ex**(1) and *ed*(1). If you don't wish to learn about this yet, you can disable this more general facility by putting the

:se nomagicCR

command in EXINIT in your environment.

The command **G**, when preceded by a number will position the cursor at that line in the file. Thus **1G** will move the cursor to the first line of the file. If you give **G** no count, then the cursor moves to the end of the file.

If you are near the end of the file and the last line is not at the bottom of the screen, the editor will place only the character '~' on each remaining line. This indicates that the last line in the file is on the screen; that is, the '~' lines are beyond the end of the file.

You can determine the state of the file you are editing by typing a **^G** command. The editor will show you the name of the file you are editing, the number of the current line, the number of lines in the buffer, and the percentage of the way through the buffer the cursor is located. Try doing this now and remember the number of the line you are on. Give a **G** command to get to the end and then another **G** command to get back where you were.

You can also get back to a previous position by using the command " (two back quotes). This is often more convenient than **G** because it requires no advance preparation. Try giving a **G** or a

search with / or ? and then a " to get back where you started. If you accidentally hit **n** or any command which moves you far away from a context of interest, you can quickly get back by hitting ".

3.3 Moving Around on the Screen

Now try just moving the cursor around on the screen. If your terminal has arrow keys (4 or 5 keys with arrows going in each direction) try them and convince yourself that they work.[5] If you don't have working arrow keys, you can always use **h**, **j**, **k**, and **l**. Experienced users of **vi** prefer these keys to arrow keys, because they are usually right underneath their fingers.

Hit the + key. Each time you do, notice that the cursor advances to the next line in the file, at the first non-white position on the line. The - key is like + but goes the other way. These are very common keys for moving the cursor up and down lines in the file. Notice that if the cursor goes off the bottom or top of the screen when using these keys, then the text will scroll down (and up if possible) to bring a line at a time into view. The RETURN key has the same effect as the + key.

The **vi** editor also has commands to take you to the top, middle and bottom of the screen. The **H** command will take you to the top line (home) on the screen. Try preceding it with a number as in **3H**. This will take you to the third line on the screen. Many **vi** commands take preceding numbers and do interesting things with them. Try **M**, which takes you to the middle line on the screen, and **L**, which takes you to the last line on the screen. The **L** command also takes counts, the **5L** command will take you to the fifth line from the bottom.

There are two control characters that move the cursor up or down a line, but keep it in the same column. The ^N causes the cursor to move to the same column of the next line. To move to the same column of the previous line use the ^P command.

3.4 Moving Within a Line

Now try picking a word on some line on the screen, not the first word on the line. Move the cursor (using RETURN and -) to be on the line where the word is. Try hitting the **w** key. This will advance the cursor to the next word on the line. Try hitting the **b** key to back up words in the line. Also try the **e** key which advances you to the end of the current word rather than to the beginning of the next word. Also try SPACE (the space bar) which moves the cursor right one character and the BS key (backspace or^H) which moves left one character. The **h** key works as ^H does and is useful if you don't have a BS key. Also, as noted above, the **l** key will move the cursor to the right.

If the line has punctuation in it you may have noticed that the **w** and **b** keys stop at each group of punctuation. You can also go backward and forward without stopping at punctuation by using **W** and **B** rather than the lower case equivalents. Think of these as bigger words. Try these on a few lines with punctuation to see how they differ from the lower case **w** and **b**.

The word keys wrap around the end of line, rather than stopping at the end. Try moving to a word on a line below where you are by repeatedly hitting **w**.

3.5 Summary

BS	3.4	Move cursor one position to the left
SPACE	3.4	Move cursor one position to the right
^B	3.1	Move backward to previous page
^D	3.1	Scroll down in the file

5. On certain terminals using v2 editors, they won't.

^E	3.1	Expose another line at bottom of screen (V3)
^F	3.1	Move forward to next page
^G	3.2	Determine state of file (file name, current line number, number of lines in the buffer, and per-cent way through the buffer)
^H	3.4	Move cursor one space to the left
^N	3.3	Move cursor to next line, same column
^P	3.3	Move cursor to previous line, same column
^U	3.1	Scroll up in the file
^Y	3.1	Expose another line at the top of screen (V3)
+	3.3	Advance cursor to beginning of next line
-	3.3	Move cursor to beginning of previous line
/	3.2	Search forward for a character string
?	3.2	Search backward for a character string
B	3.4	Move cursor backward a word, ignoring punctuation
G	3.2	Move cursor to specified line or to last line if default
H	3.3	Move cursor to top line (home) on screen
M	3.3	Move cursor to middle line on screen
L	3.3	Move cursor to last line on screen
W	3.4	Move cursor forward a word, ignoring punctuation
b	3.4	Move cursor backward a word.
e	3.4	Move cursor to end of current word
h	2.4	Moves cursor to the left
j	2.4	Moves cursor down (in same column)
k	2.4	Moves cursor up (in same column)
l	2.4	Moves cursor to the right
n	3.2	Repeat scan for next instance of / or ? pattern
w	3.4	Move cursor forward a word

3.6 The view Editor

If you want to use the editor to look at a file, rather than to make changes, invoke it as **view** instead of **vi**. This will set the *readonly* option which will prevent you from accidently overwriting the file.

4. Making Simple Changes

4.1 Inserting

One of the most useful commands is the **i** (insert) command. After you type **i**, everything you type until you hit ESC is inserted into the file. Try this now; position yourself on some word in the file and try inserting text before this word. If you are on a dumb terminal it will seem that some of the characters in your line have been overwritten, but they will reappear when you hit ESC.

Now try finding a word which can, but does not, end in an 's'. Position yourself at that word and type **e** (move to end of word), **a** (append), and 'sESC' (terminate the textual insert). This sequence of commands can be used to pluralize a word.

Try inserting and appending a few times to make sure you understand how this works:

- **i** places text to the left of the cursor
- **a** places text to the right of the cursor.

Many related editor commands are invoked by the same letter key and differ only in that one is given by a lower case key and the other is given by an upper case key. In these cases, the upper case key often differs from the lower case key in its sense of direction, with the upper case key working backward and/or up, while the lower case key moves forward and/or down.

It is often the case that you want to add new lines to the file you are editing, before or after some specific line in the file. Find a line where this makes sense and then give the **o** command to create a new line after the line you are on, or the **O** command to create a new line before the line you are on. After you create a new line in this way, the text you type up to an ESC is inserted on the new line.

Whenever you are typing in text, you can give many lines of input or just a few characters. To type in more than one line of text, hit a RETURN at the middle of your input. A new line will be created for text, and you can continue to type. If you are on a slow and dumb terminal the editor may choose to wait to redraw the tail of the screen, and will let you type over the existing screen lines. This avoids the lengthy delay which would occur if the editor attempted to keep the tail of the screen always up to date. The tail of the screen will be fixed and the missing lines will reappear when you hit ESC.

While inserting new text, you can use the characters normally used at the system command level (usually **^H** or **#**) to backspace over the last character typed, and the character used to kill input lines (usually **@**, **^X**, or **^U**) can be used to erase the input you have typed on the current line. In fact, the **^H** (backspace) always works to erase the last input character, regardless of what your erase character is. The **^W** will erase a whole word and leave you after the space following the previous word. It is useful for quickly backing up in an insert. The following conditions should be noted:

- When you backspace during an insertion, the characters you backspace over are not erased; the cursor moves backward, and the characters remain on the display. This is often useful if you are planning to type in something similar. In any case the characters disappear when you hit ESC. If you want to get rid of them immediately, hit an ESC and then **a** again.
- You cannot erase characters which you did not insert, and you cannot backspace around the end of a line. If you need to back up to the previous line to make a correction, just hit ESC and move the cursor back to the previous line. After making the correction you can return to where you were and use the insert or append command again.

4.2 Making Small Corrections

You can make small corrections in existing text quite easily. Find a single character which is wrong or just pick any character. Use the arrow keys to find the character, or get near the character with the word motion keys and then either backspace (hit the BS key, **^H**, or just **h**) or SPACE (using the space bar) until the cursor is on the character which is wrong. If the character is not needed then hit the **x** key; this deletes the character from the file. It is analogous to the way you **x** out characters when you make mistakes on a typewriter.

If the character is incorrect, you can replace it with the correct character by giving the **r***c* command, where *c* is the correct character. If the character which is incorrect should be replaced by more than one character, give the command

s*string*ESC

which substitutes a string of characters, and ends with ESC. If there are a small number of characters which are wrong you can precede **s** with a count of the number of characters to be replaced. Counts are also useful with **x** to specify the number of characters to be deleted.

4.3 Making Corrections With Operators

You already know almost enough to make changes at a higher level. All you need to know now is that the **d** key acts as a delete operator and the **c** key acts as a change operator.

- The **dw** command deletes a following word.
- The **db** command deletes a preceding word. namely the preceding word.
- The **d**SPACE command deletes a single character, and is equivalent to the **x** command.
- The **cw** command changes the text of a single word. It is followed with replacement text ending with an ESC. Find a word which you can change to another and try this now. Notice that the end of the text to be changed is marked with the character '$' so that you can see this mark as you are typing in the new text material.

The **.** command repeats the last command which made a change.

4.4 Operating on Lines

It is often the case that you want to operate on lines. Find a line you want to delete and type **dd**, the **d** operator twice. This will delete the line. If you are on a dumb terminal, the editor may erase the line on the screen, replacing it with a line with only an @ on it. This line does not correspond to any line in your file, but only acts as a place holder. It helps to avoid a lengthy redraw of the rest of the screen which would be necessary to close up the hole created by the deletion on a terminal without a delete line capability.

Try repeating the **c** operator twice (**cc**); this will change a whole line, erasing its previous contents and replacing them with text you type up to an ESC. The command **S** is a convenient synonym for **cc**, by analogy with **s**. Think of **S** as a substitute on lines, while **s** is a substitute on characters.

You can delete or change more than one line by preceding the **dd** or **cc** with a count (**5dd** deletes 5 lines). You can also give a command like **dL** to delete all the lines up to and including the last line on the screen or **d3L** to delete through the third from the bottom line. One subtle point here involves using the **/** (search) after a **d**. This will normally delete characters from the current position to the point of the match. If you desire to delete whole lines including the two points, give the pattern as **/pat/+0**, a line address.
Note: The **ex** editor lets you know when you change a large number of lines so that you can see the extent of the change. It will also always tell you when a change you make affects text which you cannot see.

4.5 Undoing

Suppose the last change you made was incorrect; you could use the insert, delete, and append commands to put the correct material back. However, since it is often the case that we regret a change or make a change incorrectly, the editor provides a **u** (undo) command to reverse the last change you made. Try this a few times, and give it twice in a row to notice that a **u** also undoes a **u**.

The undo command lets you reverse only a single change. After you make a number of changes to a line, you may decide that you would rather have the original state of the line back. The **U** command restores the current line to the state before you started changing it.

You can recover text which you deleted, even if undo will not bring it back; see subpart 7.3 on recovering lost text.

4.6 Summary

SPACE	4.2	Advance cursor one position to the right
^H	4.1	Move cursor one space to the left
^W	4.1	Erase a word during an insert
erase	4.1	Erase a character during an insert (usually ^H or #)
kill	4.1	Kill the insert on this line (usually @, ^X, or ^U)
.	4.3	Repeat the last change command
O	4.1	Opens and inputs new lines above the current line
S	4.4	Substitute on lines
U	4.5	Undoes the changes made to the current line
a	4.1	Append text after the cursor
c	4.3	Change the specified object (word) to the following text
d	4.3	Delete the specified object (word, space, etc.)
i	4.1	Insert text before the cursor
o	4.1	Opens and inputs new lines below the current line
r	4.2	Replace a character
s	4.2	Replace a character with a string
u	4.5	Undoes the last change
x	4.2	Delete a character
cc	4.4	Change a whole line
dd	4.4	Delete a line

5. Moving About, Rearranging, and Duplicating Text

5.1 Low Level Character Motions

Move the cursor to a line where there is a punctuation or a bracketing character such as a parenthesis or a comma or period. Try the command **f***x* where *x* is the character sought. This command finds the next *x* character to the right of the cursor in the current line. Try then hitting a ;, which finds the next instance of the same character. By using the **f** command and then a sequence of ;'s, you can often get to a particular place in a line much faster than with a sequence of word motions or SPACEs. There is also an **F** command, which is like **f**, but searches backward. The ; command repeats **F** also.

When operating on the text in a line, it is often desirable to delete characters including the first instance of a character. Try **df***x* for some *x* and notice that the *x* character is deleted. Undo this with **u** and then try **dt***x* (the **t** stands for "to") to delete up to the next *x*, but not the *x*. The command **T** is the reverse of **t**.

When working with the text of a single line, an ↑ moves the cursor to the first non-white position on the line, and a **$** moves it to the end of the line. Thus, **$a** will append new text at the end of the current line.

Your file may have tab characters (^I) in it. These characters are represented as a number of spaces expanding to a tab stop, where tab stops are every eight positions by default. Tab stops are settable by a command of the form

:se ts= *x*CR

where *x* is four to set tab stops every four columns. The tab stop setting has an effect on screen representation within the editor. When the cursor is at a tab, it sits on the last of the several spaces which represent that tab. Try moving the cursor back and forth over tabs so you understand how this works.

On rare occasions, your file may have non-printing characters in it. These characters are displayed in the same way they are represented in this manual; i.e., with a 2-character code, the first character of which is the '^' character. On the screen non-printing characters resemble a '^' character adjacent to another. Spacing or backspacing over the character reveals that the two characters are, like the spaces representing a tab character, a single character.

The editor sometimes discards control characters, depending on the character and the setting of the *beautify* option, if you attempt to insert them in your file. You can get a control character in the file by beginning an insert and then typing a ^V before the control character. The ^V quotes the following character causing it to be inserted directly into the file.

5.2 Higher Level Text Objects

In working with a document it is often advantageous to work in terms of sentences, paragraphs, and sections.

- The (and) operations move to the beginning of the previous and next sentence, respectively. Thus the **d)** command will delete the rest of the current sentence. The **d(** command will:

  delete the previous sentence if you are at the beginning of the current sentence
  or
  delete the current sentence up to where you are if you are not at the beginning of the current sentence.

- A sentence is defined to end at a ., !, or ? which is followed by either the end of a line or two spaces. Any number of),], ", and ' closing characters may appear after the ., !, or ?, and before the spaces or end of line.

- The { and } operations move over paragraphs.

- The [[and]] operations move over sections. They require the operation character to be doubled because they can move the cursor from where it currently is. While it is easy to get back with the '' command, these commands would still be frustrating if they were easy to hit accidentally.

- A paragraph begins after each empty line and also at each of a set of paragraph macros specified by the pairs of characters in the definition of the string valued option *paragraphs*. The default setting for this option defines the paragraph macros of the -ms and -mm macro packages, i.e., the .IP, .LP, .PP and .QP, .P and .LI macros. You can easily change or extend this set of macros by assigning a different string to the *paragraphs* option in your EXINIT. See subpart 7.2 for details. The **.bp** request is also considered to start a paragraph. Each paragraph boundary is also a sentence boundary. The sentence and paragraph commands can be given counts to operate over groups of sentences and paragraphs.

- Sections in the editor begin after each macro in the *sections* option, normally, .NH, .SH, .H and .HU, and each line with a form feed ^L in the first column. Section boundaries are always line and paragraph boundaries.

Try experimenting with the sentence and paragraph commands until you are sure how they work. If you have a large document, try looking at it using the section commands. Section commands interpret a preceding count as a different window size in which to redraw the screen at the new location, and this window size is the base size for newly drawn windows until another size is specified. This is very useful if you are on a slow terminal and are looking for a particular section. You can give the first section command and a small count to see each successive section heading in a small window.

5.3 Rearranging and Duplicating Text

The editor has a single unnamed buffer where the last deleted or changed text is saved, and a set of named buffers **a-z** which you can use to save copies of text and to move text around in your file and between files.

The **y** operator yanks a copy of the object which follows into the unnamed buffer. If preceded by a buffer name, "*x*y, where *x* is replaced by a letter **a-z**, it places the text in the named buffer. The text can then be put back in the file with the commands **p** and **P**; the **p** command puts the text after or below the cursor, while **P** puts the text before or above the cursor.

If the text which you yank forms a part of a line or is an object such as a sentence which partially spans more than one line, then when you put the text back, it will be placed after the cursor (or before if you use **P**). If the yanked text forms whole lines, they will be put back as whole lines without changing the current line. In this case, the put acts much like an **o** or **O** command (sub-part 4.1).

Try the **YP** command. This makes a copy of the current line and leaves you on this copy, which is placed before the current line. The command **Y** is a convenient abbreviation for **yy**. The command **Yp** will also make a copy of the current line and place it after the current line. You can give **Y** a count of lines to yank, and thus duplicate several lines; try **3YP**.

To move text within the buffer, you need to delete it in one place and put it back in another. You can precede a delete operation by the name of a buffer in which the text is to be stored as in **"a5dd** deleting 5 lines into the named buffer *a*. You can then move the cursor to the eventual resting place of these lines and do an **"ap** or **"aP** to put them back. In fact, you can switch and edit another file before you put the lines back by giving a command of the form

:e *name*CR

where *name* is the name of the other file you want to edit. You will have to write back the contents of the current editor buffer (or discard them) if you have made changes before the editor will let you switch to the other file. An ordinary delete command saves the text in the unnamed buffer so that an ordinary put can move it elsewhere. However, the unnamed buffer is lost when you change files; so to move text from one file to another, you should use a named buffer.

5.4 Summary

^I	5.1	Tab, add spaces up to next tab stop
^L	5.2	Form feed
^V	5.1	Quote the following character
↑	5.1	Move cursor to first non-white position on line
$	5.1	Move cursor to end of line
)	5.2	Balance of sentence forward
}	5.2	Move cursor forward over paragraph operator
]]	5.2	Move cursor forward over section operator
(	5.2	Balance of sentence backward
{	5.2	Move cursor backward over paragraph operator
[[	5.2	Move cursor backward over section operator
;	5.1	Find next instance of same character found with the **f***x* or **F***x*
d	5.1	Delete the specified object (character, word, space)

f*x*	5.1	Find the first *x* character to right of the cursor
p	5.3	Put text back, after cursor or below current line
u	5.1	Undo the last change
y	5.3	Yank operator, for copies and moves
t*x*	5.1	To *x* forward, for operators
F*x*	5.1	Find the first *x* character to left of the cursor
P	5.3	Put text back, before cursor or above current line
T*x*	5.1	Up to *x* backward in line

6. High-Level Commands

6.1 Writing, Quitting, and Editing New Files

So far we have seen how to enter the **vi** editor and to write out our file using either **ZZ** or **:w**CR commands. The first command exits from the editor (writing if changes were made), the second command writes and stays in the editor.

If you have changed the editor copy of the file but do not wish to save your changes either because you messed up the file or decided that the changes are not an improvement, then you can give the command

:q!CR

to quit from the editor without writing the changes. You can also reedit the same file (start over) by giving the command

:e!CR

These commands should be used only rarely and with caution, since it is not possible to recover the changes you have made after you discard them in this manner.

You can edit a different file without leaving the editor by giving the command

:e *name*CR.

If you have not written out your file before you try to do this, then the editor will tell you this and delay editing the other file. You can then give the command

:wCR

(to save your changes) and then the

:e *name*CR

command again or carefully give the command

:e! *name*CR

which edits the other file discarding the changes you have made to the current file. To have the editor automatically save changes, include *set autowrite* in your EXINIT and use **:n** instead of **:e**.

6.2 Escaping to a Shell

You can get to a shell to execute a single command by giving a **vi** command of the form

:!*cmd*CR

The system will run the single command (**cmd**); and when finished, the editor will ask you to hit a RETURN to continue. When you have finished looking at the output on the screen, you should hit RETURN; and the editor will clear the screen and redraw it. You can then continue editing. You can also give another : command when it asks you for a RETURN; in this case the screen will not be

redrawn.

If you wish to execute more than one command in the shell, then you can give the command

:shCR

This will give you a new shell. When finished with the shell, end it by typing a **^D**. The editor will clear the screen and continue.

6.3 Marking and Returning

The command " returns you to the previous place after a motion of the cursor by a command such as /, ?, or G. You can also mark lines in the file with single letter tags and return to these marks later by naming the tags. Try marking the current line with the command **m***x*, where you should pick some letter for *x*, such as an 'a'. Then move the cursor to a different line (any way you like) and hit **'a**. The cursor will return to the place which you marked. Marks last only until you edit another file.

When using operators such as **d** and referring to marked lines, it is often desirable to delete whole lines rather than deleting to the exact position in the line marked by **m**. In this case you can use the form '*x* rather than '*x*. Used without an operator, '*x* will move to the first non-white character of the marked line; similarly, " moves to the first non-white character of the line containing the previous context mark ".

6.4 Adjusting the Screen

If the screen image is messed up because of a transmission error to your terminal or because some program other than the editor wrote output to your terminal, you can hit a **^L**, the ASCII form-feed character, to cause the screen to be refreshed.

On a dumb terminal, if there are @ lines in the middle of the screen as a result of line deletion, you may get rid of these lines by typing **^R** to cause the editor to retype the screen, closing up these holes.

If you wish to place a certain line on the screen at the top, middle, or bottom of the screen, you can position the cursor to that line and then give a **z** command. You should follow the **z** command with a RETURN if you want the line to appear at the top of the window, a **.** if you want it at the center, or a **-** if you want it at the bottom.

7. Special Topics

7.1 Editing on Slow Terminals

When you are on a slow terminal, it is important to limit the amount of output which is generated to your screen so that you will not suffer long delays waiting for the screen to be refreshed. We have already pointed out how the editor optimizes the updating of the screen during insertions on dumb terminals to limit the delays and how the editor erases lines to @ when they are deleted on dumb terminals.

The use of the slow terminal insertion mode is controlled by the *slowopen* option. You can force the editor to use this mode even on faster terminals by giving the command **:se slow**CR. If your system is sluggish, this helps lessen the amount of output coming to your terminal. You can disable this option by **:se noslow**CR.

The editor can simulate an intelligent terminal on a dumb one. Try giving the command **:se redraw**CR. This simulation generates a great deal of output and is generally tolerable only on lightly loaded systems and fast terminals. You can disable this with the **:se noredraw**CR command.

The editor also makes editing more pleasant at low speed by starting editing in a small window and letting the window expand as you edit. This works particularly well on intelligent terminals. The editor can expand the window easily when you insert in the middle of the screen on these terminals. If possible, try the editor on an intelligent terminal to see how this works.

You can control the size of the window which is redrawn each time the screen is cleared by giving window sizes as argument to the commands which cause large screen motions:

: / ? [[ ]] ' '

Thus if you are searching for a particular instance of common string in a file you can precede the first search command by a small number, such as 3, and the editor will draw three line windows around each instance of the string which it locates.

You can easily expand or contract the window and place the current line as you choose by giving a number with the **z** command (after the **z** and before the following RETURN, . or -). Thus the command **z5.** redraws the screen with the current line in the center of a five line window.

Note: The command **5z.** has an entirely different effect placing line 5 in the center of a new window.

If the editor is redrawing or otherwise updating large portions of the display, you can interrupt this updating by hitting a DEL or RUB as usual. If you do this you may partially confuse the editor about what is displayed on the screen. You can still edit the text on the screen if you wish; clear up the confusion by hitting a ^L; or move or search again, ignoring the current state of the display.

See subpart 9.8 on *open* mode for another way to use the **vi** command set on slow terminals.

7.2 Options, Set, and Editor Startup Files

The editor has a set of options, some of which have been mentioned above. The most useful options are given in the following table.

OPTION	DEFAULT	DESCRIPTION
autoindent	noai	Supply indentation automatically
autowrite	noaw	Automatic write before :n, :ta, ^↑, and !
ignorecase	noic	Ignore case in searching
lisp	nolisp	({) } commands deal with S-expressions
list	nolist	Tabs print as ^I; end of lines marked with $
magic	nomagic	The characters [and * are special in scans
number	nonu	Lines are displayed prefixed with line numbers
paragraphs	para=IPLPPPQPbpP LI	Macro names which start paragraphs
redraw	nore	Simulate a smart terminal on a dumb one
sections	sect=NHSHH HU	Macro names which start new sections
shiftwidth	sw=8	Shift distance for <, > and input ^D and ^T
showmatch	nosm	Show matching (or { as) or } is typed
slowopen	slow	Postpone display updates during inserts
term	dumb	The kind of terminal you are using

The options are of three kinds: numeric, string, and toggle. You can set numeric and string options by a statement of the form

set *opt*=*val*

and toggle options can be set or unset by statements of one of the forms

set *opt*
set no*opt*

These statements can be placed in your EXINIT in your environment or given while you are running **vi** by preceding them with a : and following them with a CR.

You can get a list of all options that you have changed with the command

:**set**CR

or the value of a single option by the command

:**set** *opt*?CR

A list of all possible options and their values is generated by

:**set all**CR

Set can be abbreviated **se**. Multiple options can be placed on one line, for instance:

:**se ai aw nu**CR

Options set by the **set** command last only while you stay in the editor. It is common to want to have certain options set whenever you use the editor. This can be accomplished by creating a list of **ex** commands[6] which are to be run every time you start up **ex**, **edit**, or **vi**. A typical list includes a **set** command and possibly a few **map** commands (on V3 editors). Since it is advisable to get these commands on one line, they can be separated with the | character; for example:

set ai aw terse|**map** @ dd|**map** # x

which establishes the **set** command options *autoindent*, *autowrite*, *terse*, makes @ delete a line (the first **map**), and makes # delete a character (the second **map**) (see subpart 7.9 for a description of the **map** command, which works only in version 3). This string should be placed in the variable EXINIT in your environment. Using the shell, put these lines in the file *.profile* in your home or working directory:

EXINIT=**set** ai aw terse|**map** @ dd|**map** # x
export EXINIT

Or put the line in the file *exrc* in your home directory.

set ai aw terse|**map** @ dd|**map** # x

Of course, the particulars of the line would depend on which options you wanted to set.

7.3 Recovering Lost Lines

You might have a serious problem if you delete a number of lines and then regret that they were deleted. Despair not, the editor saves the last nine deleted blocks of text in a set of numbered registers 1 through 9. You can get the *n*th previous deleted text back in your file by the command "*n***p**. The " here says that a buffer name is to follow, *n* is the number of the buffer you wish to try (use the number 1 for now), and **p** is the put command which puts text in the buffer after the cursor. If this doesn't bring back the text you wanted, hit **u** to undo this and then . (period) to repeat the put command. In general the . command will repeat the last change. As a special case, when the last command refers to a numbered text buffer, the . command increments the number of the buffer before repeating the command. Thus a sequence of the form

. "**1pu.u.u**

---

6. All commands which start with : are **ex** commands.

will, if repeated long enough, show you all the deleted text which has been saved for you. You can omit the **u** commands here to gather up all this text in the buffer or stop after any . command to keep just the then-recovered text. The command **P** can also be used rather than **p** to put the recovered text before rather than after the cursor.

7.4 Recovering Lost Files

If the system crashes, you can recover the work you were doing to within a few changes. You will normally receive mail when you next login giving you the name of the file which has been saved for you. You should then change to the directory where you were when the system crashed and give a command of the form:

% **vi -r** *name*

replacing *name* with the name of the file which you were editing. This will recover your work to a point near where you left off.[7]

You can get a listing of the files which are saved for you by giving the command:

% **vi -r**

If there is more than one instance of a particular file saved, the editor gives you the newest instance each time you recover it. You can thus get an older saved copy back by first recovering the newer copies.

For this feature to work, **vi** must be correctly installed by a superuser on your system, and the **mail** program must exist to receive mail. The invocation "**vi -r**" will not always list all saved files, but they can be recovered even if they are not listed.

7.5 Continuous Text Input

When you are typing in large amounts of text, it is convenient to have lines broken near the right margin automatically. You can cause this to happen by giving the command

:se wm=10CRCR

This causes all lines to be broken at a space at least ten columns from the righthand edge of the screen.[8]

If the editor breaks an input line and you wish to put it back together, you can tell it to join the lines with **J**. You can give **J** a count of the number of lines to be joined (as in **3J** to join 3 lines). The editor supplies white space, if appropriate, at the juncture of the joined lines, and leaves the cursor at this white space. You can kill the white space with **x** if you don't want it.

7.6 Features for Editing Programs

The editor has a number of commands for editing programs. The thing that most distinguishes editing of programs from editing of text is the desirability of maintaining an indented structure to the body of the program. The editor has an *autoindent* facility to help generate correctly indented programs.

To enable this facility, you can give the command **:se ai**CR. Now try opening a new line with **o** and then type some characters on the line after a few tabs. If you now start another line, notice

---

7. In rare cases, some of the lines of the file may be lost. The editor will give you the numbers of these lines and the text of the lines will be replaced by the string 'LOST'. These lines will almost always be among the last few which you changed. You can either choose to discard the changes which you made (if they are easy to remake) or to replace the few lost lines by hand.

8. This feature is not available on some V2 editors. In V2 editors where it is available, the break can only occur to the right of the specified boundary instead of to the left.

that the editor supplies white space at the beginning of the line to line it up with the previous line. You cannot backspace over this indentation, but you can use ^D key to backtab over the supplied indentation.

Each time you type ^D you back up one position, normally to an 8-column boundary. This amount is settable; the editor has an option called *shiftwidth* which you can set to change this value. Try giving the command

:see sw=4CR

and then experimenting with autoindent again.

For shifting lines in the program left and right, there are operators < and >. These shift the lines you specify right or left by one *shiftwidth*. Try < < and > > which shift one line left or right, and <L and >L shifting the rest of the display left and right.

If you have a complicated expression and wish to see how the parentheses match, put the cursor at a left or right parenthesis and hit %. This will show you the matching parenthesis. This works also for braces ({}), and brackets ([]).

If you are editing C programs, you can use the [[and]] keys to advance or retreat to a line starting with a {, i.e., a function declaration at a time. When]] is used with an operator, it stops after a line which starts with }; this is sometimes useful with **y]]**.

7.7 Filtering Portions of the Buffer

You can run system commands over portions of the buffer using the operator!. You can use this to sort lines in the buffer or to reformat portions of the buffer with a pretty-printer. Try typing in a list of random words, one per line and ending them with a blank line. Back up to the beginning of the list and then give the command **!}sort**CR. This says to sort the next paragraph of material, and the blank line ends a paragraph.

7.8 Commands for Editing LISP[9]

If you are editing a **LISP** program, you should set the option *lisp* by doing

:se lispCR

This changes the (and) commands to move backward and forward over s-expressions. The { and } commands are like (and) but don't stop at atoms. These can be used to skip to the next list or through a comment quickly.

The *autoindent* option works differently for **LISP** supplying indent to align at the first argument to the last open list. If there is no such argument, then the indent is two spaces more than the last level.

There is another option which is useful for typing in **LISP**, the *showmatch* option. Try setting it with **:se sm**CR and then try typing a (some words and then a). Notice that the cursor shows the position of the (which matches the) briefly. This happens only if the matching (is on the screen, and the cursor stays there for at most one second.

The editor also has an operator to realign existing lines as though they had been typed in with *lisp* and *autoindent* set. This is the = operator. Try the command **=%** at the beginning of a function. This will realign all the lines of the function declaration.

When you are editing **LISP**, the [[and]] advance and retreat to lines beginning with a (and are

---

9. The **LISP** features are not available on some V2 editors due to memory constraints.

useful for dealing with entire function definitions.

7.9 Macros[10]

The **vi** editor has a macro facility, which lets you set up a macro so that when you enter a single keystroke, the editor will act as though you had entered some longer sequence of keystrokes. You can set up a macro if you find yourself typing the same sequence of commands (keystrokes) repeatedly.

Briefly, there are two types of macros:

a) One type is where you put the macro body in a buffer register, such as *x*. You can then type @x to invoke the macro. The @ may be followed by another @ to repeat the last macro.

b) The other type is where you can use the map command from **vi** (typically in your EXINIT) with a command of the form:

:map *lhs rhs*CR

mapping *lhs* into *rhs*. There are restrictions: *lhs* should be one keystroke (either one character or one function key) since it must be entered within 1 second (unless *notimeout* is set, in which case you can type it as slowly as you wish, and **vi** will wait for you to finish it before it echoes anything). The *lhs* can be no longer than ten characters, the *rhs* no longer than 100. To get a space, tab or newline into *lhs* or *rhs* you should escape them with a ^V (it may be necessary to double the ^V if the map command is given inside **vi** rather than in **ex**). Spaces and tabs inside the *rhs* need not be escaped.

Thus to make the **q** key write and exit the editor, you can give the command

:map q :wq^V^VCR CR

which means that whenever you type **q**, it will be as though you had typed the four characters **:wq**CR. A ^V is needed because without it the CR would end the : command rather than becoming part of the *map* definition. There are two ^V's because from within **vi**, two ^V's must be typed to get on. The first CR is part of the *rhs*, the second terminates the : command.

Macros can be deleted with

unmap *lhs*

If the *lhs* of a macro is "#0" through "#9", this maps the particular function key instead of the 2-character "#" sequence. So that terminals without function keys can access such definitions, the form "#x" will mean function key *x* on all terminals (and need not be typed within 1 second). The character "#" can be changed by using a macro in the usual way:

:map **^V^V^I #**

to use tab, for example. This will not affect the *map* command, which still uses #, but just the invocation from visual mode.

The undo command reverses an entire macro call as a unit if it made any changes.

Placing a "!" after the word map causes the mapping to apply to input mode rather than command mode. Thus, to arrange for **^T** to be the same as four spaces in input mode, you can type:

---

10. The macro feature is available only in version 3 editors.

:map ^T ^V␣␣␣␣

where ␣ is a blank. The ^V is necessary to prevent the blanks from being taken as white space between the *lhs* and *rhs*.

8. Word Abbreviations[11]

A feature similar to macros in input mode is word abbreviation. This allows you to type a short word and have it expanded into a longer word or words. The commands are

:abbreviate
and
:unabbreviate

or

:ab
and
:una

and have the same syntax as **:map**. For example:

:ab eecs Electrical Engineering and Computer Sciences

causes the word 'eecs' to always be changed into the phrase 'Electrical Engineering and Computer Sciences'. Word abbreviation is different from macros in that only whole words are affected. If 'eecs' were typed as part of a larger word, it would be left alone. Also, the partial word is echoed as it is typed. There is no need for an abbreviation to be a single keystroke as it should be with a macro.

8.1 Abbreviations

The editor has a number of short commands abbreviate longer commands which we have introduced here. You can find these commands easily on the quick reference card. They often save a bit of typing and you can learn them as convenient.

9. Additional Information

9.1 Line Representation in the Display

The editor folds long logical lines onto many physical lines in the display. Commands which advance lines advance logical lines and will skip over all the segments of a line in one motion. The | command moves the cursor to a specific column and may be useful for getting near the middle of a long line to split it in half. Try **80|** on a line which is more than 80 columns long.[12]

The editor puts only full lines on the display; if there is not enough room on the display to fit a logical line, the editor leaves the physical line empty, placing only an @ on the line as a place holder. When you delete lines on a dumb terminal, the editor will often clear just the lines to @ to save time (rather than rewriting the rest of the screen). You can always maximize the information on the screen by giving the **^R** command.

If you wish, you can have the editor place line numbers before each line on the display. Give the command

---

11. Version 3 only.

12. You can make long lines very easily by using **J** to join together short lines.

:se nuCR

to enable this, and the command

:se nonuCR

to turn it off. You can have tabs represented as **^I** and the ends of lines indicated with '**$**' by giving the command

:se listCR

The following command removes the display of tabs and ends of lines:

:se nolistCR

Lines consisting of only the '~' character are displayed when the last line in the file is in the middle of the screen. These represent physical lines which are past the logical end of file.

9.2 Counts

Most **vi** commands will use a preceding count to affect their behavior in some way. The following table gives the common ways in which the counts are used:

new window size	**: / ? [[ ]] ' '**
scroll amount	**^D ^U**
line/column number	**z G |**
repeat effect	most of the rest

The editor maintains a notion of the current default window size. On terminals which run at speeds greater than 1200 baud the editor uses the full terminal screen. On terminals which are slower than 1200 baud (most dial-up lines are in this group) the editor uses 8 lines as the default window size. At 1200 baud the default is 16 lines.

This size is the size used when the editor clears and refills the screen after a search or other motion moves far from the edge of the current window. Commands which take a new window size as count - all too often cause the screen to be redrawn. If you anticipate this, but do not need as large a window as you are currently using, you may wish to change the screen size by specifying the new size before these commands. In any case, the number of lines used on the screen will expand if you move off the top with a **-** or similar command, or off the bottom with a command such as RETURN or **^D**. The window will revert to the last specified size the next time it is cleared and refilled.[13]

The scroll commands **^D** and **^U** likewise remember the amount of scroll last specified, using half the basic window size initially. The simple insert commands use a count to specify a repetition of the inserted text. Thus

10a +----ESC

will insert a grid-like string of text. A few commands also use a preceding count as a line or column number.

Except for a few commands which ignore any counts (such as **^R**), the rest of the editor commands use a count to indicate a simple repetition of their effect. Thus **5w** advances five words on the current line, while **5**RETURN advances five lines. A very useful instance of a count as a repetition is a count given to the **.** command, which repeats the last changing command. If you do **dw** and the **3.**, you will delete first one and then three words. You can then delete two more words with the **2.** command.

---

13. But not by a **^L** which just redraws the screen as it is.

9.3 More File Manipulation Commands

The following table lists the file manipulation commands which you can use when you are in **vi**.

:w	write back changes
:wq	write and quit
:x	write (if necessary) and quit (same as **ZZ**).
:e *name*	edit file *name*
:e!	re-edit, discarding changes
:e + *name*	edit, starting at end
:e +*n*	edit, starting at line *n*
:e #	edit alternate file
:w *name*	write file *name*
:w! *name*	overwrite file *name*
:*x*,*y***w** *name*	write lines *x* through *y* to *name*
:r *name*	read file *name* into buffer
:r *!cmd*	read output of *cmd* into buffer
:n	edit next file in argument list
:n!	edit next file, discarding changes to current
:n *args*	specify new argument list
:ta *tag*	edit file containing tag *tag*, at *tag*

All of these commands are followed by a CR or ESC. The most basic commands are **:w** and **:e**. A normal editing session on a single file will end with a **ZZ** command. If you are editing for a long period of time you can give **:w** commands occasionally after major amounts of editing, and then finish with a **ZZ**. When you edit more than one file, you can finish with one with a **:w** and start editing a new file by giving a **:e** command, or set *autowrite* and use **:n** <file>.

If you make changes to the editor copy of a file, but do not wish to write them back, then you must give an ! after the command you would otherwise use; this forces the editor to discard any changes you have made. **Use this carefully.**

The :e command can be given a + argument to start at the end of the file, or a +*n* argument to start at line *n*. In actuality, *n* may be any editor command not containing a space, usually a scan like +/*pat* or +?*pat*. In forming new names to the **e** command, you can use the character % which is replaced by the current file name, or the character # which is replaced by the alternate file name. The alternate file name is generally the last name you typed other than the current file. Thus, if you try to do a **:e** and get a diagnostic that you have not written into the file, you can give a **:w** command and then a **:e #** command to redo the previous **:e**.

You can write part of the buffer to a file by finding the lines that bound the range to be written using ^G, to obtain the line numbers, and giving these numbers after the : and before the **w**, separated by ,'s. You can also mark these lines with **m** and then use an address of the form '*x*,'*y* on the **w** command.

You can read another file into the buffer after the current line by using the **:r** command. You can similarly read in the output from a command, just use the **:r** *!cmd* instead of a file name.

If you wish to edit a set of files in succession, you can give all the names on the command line, and then edit each one in turn using the command **:n**. It is also possible to respecify the list of files to be edited by giving the **:n** command a list of file names, or a pattern to be expanded as you would have given it on the initial **vi** command.

If you are editing large programs, you will find the **:ta** command very useful. It utilizes a data base of function names and their locations, which can be created by programs such as **ctags**, to quickly find a function whose name you give. If the **:ta** command will require the editor to switch files, then you must **:w** or abandon any changes before switching. You can repeat the **:ta** command without any arguments to look for the same tag again. (The tag feature is not available in some v2 editors.)

9.4 More About Searching for Strings

When you are searching for strings in the file with **/** or **?**, the editor normally places you at the next or previous occurrence of the string. If you are using an operator such as **d**, **c**, or **y**, you may well wish to affect lines up to the line before the line containing the pattern. You can give a search of the form

/pat/ -n

to refer to the *n*'th line before the next line containing *pat*, or you can use + instead of - to refer to the lines after the one containing *pat*. If you do not give a line offset, the editor will affect characters up to the match place rather than whole lines; thus use "+0" to affect to the line which matches.

You can have the editor ignore the case of words in the searches it does by giving the command

:se icCR

The command

:se noicCR

turns this off.

Strings given to searches may actually be regular expressions. If you do not want or need this facility, you should

set nomagic

in your EXINIT. In this case, only the characters ↑ and **$** are special in patterns. The character \ is also then special (as it is most everywhere in the system), and may be used to get at the extended pattern-matching facility. It is also necessary to use a \ before a / in a forward scan or a ? in a backward scan. The following table gives the extended forms when **magic** is set.

↑	at beginning of pattern, matches beginning of line
$	at end of pattern, matches end of line
.	matches any character
\<	matches the beginning of a word
\>	matches the end of a word
[*str*]	matches any single character in *str*
[↑*str*]	matches any single character not in *str*
[*x*-*y*]	matches any character between *x* and *y*
*	matches any number of the preceding pattern

If you use **nomagic** mode, then the . [and * primitives are given with a preceding \.

9.5 More About Input Mode

There are a number of characters which you can use to make corrections during input mode. These are summarized in the following table.

^H	deletes the last input character
^W	deletes the last input word
erase	erase character, same as **^H**
kill	kill character, deletes the input on this line
\	escapes a following **^H**, erase, and kill
ESC	ends an insertion
DEL	interrupt an insertion, terminating it abnormally
CR	starts a new line
^D	backtabs over *autoindent*
0^D	kills all the *autoindent*
↑^D	same as **0^D**, but restores indent next line

^V quotes the next non-printing character into the file

The most usual way of making corrections to input is by typing **^H** to correct a single character, or by typing one or more **^W**'s to back over incorrect words. If you use **#** as your erase character in the normal system, it will work like **^H**.

Your system kill character, normally **@**, **^X**, or **^U** will erase all the input you have given on the current line. In general, you can neither erase input back around a line boundary nor can you erase characters which you did not insert with this insertion command. To make corrections on the previous line after a new line has been started you can hit ESC to end the insertion, move over and make the correction, and then return to where you were to continue. The command **A** which appends at the end of the current line is often useful for continuing.

If you wish to type in your erase or kill character (such as **#** or **@**) then you must precede it with a \, just as you would do at the normal system command level. A more general way of typing non-printing characters into the file is to precede them with a **^V**. The **^V** echoes as a ↑ character on which the cursor rests. This indicates that the editor expects you to type a control character. In fact you may type any character and it will be inserted into the file at that point.

Note: This is not quite true. The implementation of the editor does not allow the NULL (^@) character to appear in files. Also the LF (line feed or **^J**) character is used by the editor to separate lines in the file, so it cannot appear in the middle of a line. You can insert any other character, however, if you wait for the editor to echo the ↑ before you type the character. In fact, the editor will treat a following letter as a request for the corresponding control character. This is the only way to type **^S** or **^Q**, since the system normally uses them to suspend and resume output and never gives them to the editor to process.

If you are using *autoindent* you can backtab over the indent which it supplies by typing a **^D**. This backs up to a *shiftwidth* boundary. This works only immediately after the supplied *autoindent*.

When you are using *autoindent* you may wish to place a label at the left margin of a line. The way to do this easily is to type ↑ and then **^D**. The editor will move the cursor to the left margin for one line and restore the previous indent on the next. You can also type a **0** followed immediately by a **^D** if you wish to kill all the indent and not have it come back on the next line.

9.6 Upper Case Only Terminals

If your terminal has only upper case, you can still use **vi** by using the normal system convention for typing on such a terminal. Characters which you normally type are converted to lower case, and you can type upper case letters by preceding them with a \. The characters

{ ~ } | '

are not available on such terminals, but you can escape them as

\(↑)\!\' .

These characters are represented on the display in the same way they are typed (the \ character you give will not echo until you type another key).

9.7 Relation Between vi and ex Editors

The **vi** editor is actually one mode of editing within the editor **ex**. When you are running **vi** you can escape to the line-oriented editor of **ex** by giving the command **Q**. All of the : commands which were introduced above are available in **ex**. Likewise, most **ex** commands can be invoked from **vi** using :. Just give them without the : and follow them with a CR.

In rare instances, an internal error may occur in **vi**. In this case you will get a diagnostic and be left in the command mode of **ex**. You can then save your work and quit if you wish by giving a command **x** after the : which **ex** prompts you with, or you can reenter **vi** by giving **ex** a **vi** command.

There are a number of things which you can do more easily in **ex** than in **vi**. Systematic changes in line-oriented material are particularly easy. You can read the advanced editing documents for the editor **ed** to find out a lot more about this style of editing. Experienced users often mix their use of **ex** command mode and **vi** command mode to speed the work they are doing.

9.8 Open Mode: vi on Hard Copy Terminals and Glass tty's[14]

If you are on a hard copy terminal or a terminal which does not have a cursor which can move off the bottom line, you can still use the command set of **vi**, but in a different mode. When you give a **vi** command, the editor will tell you that it is using *open* mode. This name comes from the *open* command in **ex**, which is used to get into the same mode.

The only difference between *visual* mode and *open* mode is the way in which the text is displayed.

In *open* mode the editor uses a single line window into the file, and moving backward and forward in the file causes new lines to be displayed, always below the current line. Two commands of **vi** work differently in *open* mode: **z** and **^R**. The **z** command does not take parameters, but rather draws a window of context around the current line and then returns to the current line.

If you are on a hard copy terminal, the **^R** command will retype the current line. On such terminals, the editor normally uses two lines to represent the current line. The first line is a copy of the line as you started to edit it, and you work on the line below this line. When you delete characters, the editor types a number of \'s to show you the characters which are deleted. The editor also reprints the current line soon after such changes so that you can see what the line looks like again.

It is sometimes useful to use this mode on very slow terminals which can support **vi** in the full screen mode. You can do this by entering **ex** and using an *open* command.

---

14. Not available in all V2 editors due to memory constraints.

TABLE A

CHARACTER FUNCTIONS SUMMARY

This table shows the uses that the **vi** editor makes of each character. Characters are presented in their order in the ASCII character set: control characters first, most special characters, digits, upper case characters, and then lower case characters.

Each character is defined with a meaning it has as a command and any meaning it has during an insert. If it has meaning only as a command, then only this is discussed. Numbers in parentheses indicate where the character is discussed (an 'f' after the section number means that the character is mentioned in a footnote).

^@ Not a command character.
If typed as the first character of an insertion it is replaced with the last text inserted, and the insert terminates. Only 128 characters are saved from the last insert; if more characters have been inserted the mechanism is not available. A ^@ cannot be part of the file due to the editor implementation.
(8.5f)

^A Unused.

^B Backward window. A count specifies repetition. Two lines of continuity are kept if possible.
(3.1, 7.1, 8.2)

^C Unused.

^D As a command, it scrolls down a half-window of text. A count gives the number of (logical) lines to scroll, and is remembered for future **^D** and **^U** commands.
(3.1, 8.2)
During an insert, it backtabs over *autoindent* white space at the beginning of a line. This white space cannot be backspaced over.
(7.6, 8.5)

^E Exposes one more line below the current screen in the file, leaving the cursor where it is if possible. (Version 3 only)

^F Forward window. A count specifies repetition. Two lines of continuity are kept if possible
(3.1, 7.1, 8.2)

^G Equivalent to **:fCR**, printing the current file name, whether it has been modified, the current line number, the number of lines in the file, and the percent of the way through the file.

^H (BS) Same as **left arrow** (see **h**). During an insert, it eliminates the last input character, backing over it but not erasing it. The character remains so you can see what you typed if you wish to type something slightly different.
(4.1, 8.5)

.^I (TAB) Not a command character.
When inserted it prints as some number of spaces. When the cursor is at a tab character it rests at the last of the spaces which represent the tab. The spacing of tab stops is controlled by the *tabstop* option.
(5.1, 7.6)

^J (LF) **Down arrow**. It moves the cursor one line down in the same column. If the position does not exist, **vi** comes as close as possible to the same column. Synonyms include **j** and

^N.

^K Unused.

^L The ASCII form feed character, which causes the screen to be cleared and redrawn. It is useful after a transmission error, if characters typed by a program other than the editor scramble the screen, or after output is stopped by an interrupt.
(6.4, 8.2f)

^M (CR) A carriage return advances to the next line, to the first nonwhite position in the line. Given a count, it advances that many lines.
(2.3)
During an insert, a CR causes the insert to continue onto another line.
(4.1)

^N **Down arrow**. It moves the cursor one line down in the same column. If the position does not exist, **vi** comes as close as possible to the same column. Synonyms include **j** and **^J**.

^O Unused.

^P **Up arrow**. It moves the cursor one line up. A synonym is **k**.

^Q Not a command character.
In input mode, **^Q** quotes the next character, the same as **^V**, except that some teletype drivers will eat the **^Q** so that the editor never sees it.

^R Redraws the current screen, eliminating logical lines not corresponding to physical lines (lines with only a single **@** character on them). On hard-copy terminals in *open* mode, retypes the current line.
(6.4, 8.2, 8.8)

^S Unused.
Some teletype drivers use **^S** to suspend output until **^Q** is invoked.

^T Not a command character.
During an insert, with *autoindent* set and at the beginning of the line, it inserts *shiftwidth* white space.

^U Scrolls the screen up, inverting **^D** which scrolls down. A count gives the number of (logical) lines to scroll, and is remembered for future **^D** and **^U** commands. The previous scroll amount is common to both. On a dumb terminal, **^U** will often necessitate clearing and redrawing the screen further back in the file.
(3.1, 8.2)

^V Not a command character.
In input mode, it quotes the next character so that it is possible to insert non-printing and special characters into the file.
(5.2, 8.5)

^W Not a command character.
During an insert, it backs up as **b** would in command mode; the deleted characters remain on the display (see **^H**).
(8.5)

^X Unused.

^Y Exposes one more line above the current screen, leaving the cursor where it is if possible. There is no mnemonic value for this key; however, it is next to **^U** (which scrolls up a bunch) (Version 3 only).

^Z Unused.

^[(ESC) Cancels a partially formed command (such as a **z** when no following character has yet been given), terminates inputs on the last line (read by commands such as :, /, and ?), and ends insertions of new text into the buffer. If an ESC is given when quiescent in command state, the editor rings the bell or flashes the screen. You can thus hit ESC if you don't know what is happening until the editor rings the bell. If you do not know if you are in insert mode, you can type ESC**a** and then the material to be input; the material will be inserted correctly whether or not you were in insert mode when you started.
(2.4, 4.1, 8.5)

^\ Unused.

^] Searches for the word which is after the cursor as a tag. It is equivalent to typing **:ta**, this word, and then a CR.
(8.3)

^↑ Equivalent to **:e** #CR, returning to the previous position in the last edited file, or editing a file which you specified if you got a "No write since last change" diagnostic and do not want to have to type the file name again. You have to do a **:w** before ^↑ will work in this case. If you do not wish to write the file you should do **:e!** #CR instead.
(7.3)

^_ Unused.
Reserved as the command character for the Tektronix 4025 and 4027 terminal.

SPACE Same as **right arrow** (see **l**).

! An operator, which processes lines from the buffer with reformatting commands. Follow ! with the object to be processed, and then the command name terminated by CR. Doubling ! and preceding it by a count causes count lines to be filtered; otherwise, the count is passed on to the object after the !. Thus **2!}***fmt*CR reformats the next two paragraphs by running them through the program *fmt*.
(7.7, 8.3)
To read a file or the output of a command into the buffer use **:r**.
(8.3)
To simply execute a command use **:!**.
(8.3)

" Precedes a named buffer specification. There are named buffers **1-9** used for saving deleted text and named buffers **a-z** into which you can place text.
(5.3, 7.3)

The macro character which, when followed by a number, will substitute for a function key on terminals without function keys. In input mode, if this is your erase character, it will delete the last character you typed in input mode, and must be preceded with a \ to insert it, since it normally backs over the last input character you gave.
(6.9)

$** Moves to the end of the current line. If the **:se list**CR command is used, then the end of each line will be shown by printing a **$ after the end of the displayed text in the line. When a count is used, the cursor advances to the end of the line following the count. For example **2$** advances the cursor to the end of the following line.

% Moves to the parenthesis(()) or brace ({) which balances the parenthesis or brace at the current cursor position.

& A synonym for **:&**CR, analogous to the **ex &** command.

' When followed by a ', the cursor returns to the previous context at the beginning of a line. The previous context is set whenever the current line is moved in a nonrelative way. When followed by a letter (**a-z**), it returns to the line which was marked with this letter with an **m** command, at the first nonwhite character in the line. When used with an operator such as **d**, the operation takes place over complete lines; if you use ', the operation takes place from the exact marked place to the current cursor position within the line.
(3.2, 6.3)

(Retreats to the beginning of a sentence, or to the beginning of a LISP s-expression if the *lisp* option is set. A sentence ends at a ., !, or ? which is followed by either the end of a line or by two spaces. Any number of closing characters (),], ", and ') may appear after the ., !, or ?, and before the spaces or end of line. Sentences also begin at paragraph and section boundaries (see { and [[). A count advances that many sentences
(5.2, 7.8)

) Advances to the beginning of a sentence. A count repeats the effect. See (for the definition of a sentence.
(5.2, 7.8)

* Unused.

+ Same as CR when used as a command.

, Reverse of the last **f**, **F**, **t**, or **T** command, looking the other way in the current line. Especially useful after hitting too many ; characters. A count repeats the search.

- Retreats to the previous line at the first nonwhite character. This is the inverse of + and RETURN. If the line moved to is not on the screen, the screen is scrolled, or cleared and redrawn. If a large amount of scrolling would be required the screen is also cleared and redrawn, with the current line at the center.
(3.3)

. Repeats the last command which changed the buffer. Especially useful when deleting words or lines; you can delete some words/lines and then hit . to delete more and more words/lines. Given a count, it passes it on to the command being repeated. Thus after a **2dw**, a **3.** deletes three words.
(4.3, 7.3, 8.2, 8.4)

/ Reads a string from the last line on the screen, and scans forward for the next occurrence of this string. The search begins when you hit CR to terminate the pattern; the cursor moves to the beginning of the last line to indicate that the search is in progress; the search may be terminated with a DEL or RUB, or by backspacing when at the beginning of the bottom line, returning the cursor to its initial position. Searches normally wrap end-around to find a string anywhere in the buffer.

When used with an operator, the enclosed region is normally affected. By mentioning an offset from the line matched by the pattern, you can force whole lines to be affected. To do this, give a pattern with a closing / and then an offset +*n* or -*n*.

To include the / character in the search string, you must escape it with a preceding \. A ↑ at the beginning of the pattern forces the match to occur at the beginning of a line only; this speeds the search. A **$** at the end of the pattern forces the match to occur at the end of a line only. More extended pattern matching is available (see section 8.4). Unless you set **nomagic** in your *.exrc* file you will have to precede the characters ., [, *, and ~ in the search pattern with a \ to get them to work as you would expect.
(2.4, 3.2, 7.1, 8.2, 8.4)

0 Moves to the first character on the current line. Also used, when forming numbers, after an initial **1-9**.

1-9 Used to form numeric arguments to commands.
(3.3, 8.2)

: A prefix to a set of commands for file and option manipulation and escapes to the system. Input is given on the bottom line and terminated with a CR, and the command is then executed. You can return to where you were by hitting DEL or RUB if you hit : accidentally.
(7.2 and 8.3)

; Repeats the last single character find which used **f**, **F**, **t**, or **T**. A count iterates the basic scan.
(5.1)

< An operator which shifts lines left one *shiftwidth*, normally 8 spaces. Like all operators, it affects lines when repeated, as in **< <**. Counts are passed through to the basic object, thus **3<<** shifts three lines.
(7.6., 8.2)

= Reindents lines for LISP, as though they were typed in with *lisp* and *autoindent* set.
(7.8)

> An operator which shifts lines right one *shiftwidth*, normally 8 spaces. Affects lines when repeated as in **>>**. Counts repeat the basic object.
(7.6, 8.2)

? Scans backwards, the opposite of **/**. See the **/** description for details on scanning.
(3.2, 7.1, 8.4)

@ A macro character.
(7.9)
If this is your kill character, you must escape it with a \ to type it in during input mode, as it normally backs over the input you have given on the current line. (4.1, 4.4, 8.5)

A Appends at the end of line, a synonym for **$a**.
(8.2)

B Backs up a word, where words are composed of nonblank sequences, placing the cursor at the beginning of the word. A count repeats the effect.
(3.4)

C Changes the rest of the text on the current line; a synonym for **c$**.

D Deletes the rest of the text on the current line; a synonym for **d$**:

E Moves forward to the end of a word, defined as blanks and nonblanks, like **B** and **W**. A count repeats the effect.

F Finds a single following character, backwards in the current line. A count repeats this search that many times.
(5.1)

G Goes to the line number given as preceding argument, or the end of the file if no preceding count is given. The screen is redrawn with the new current line in the center if necessary.
(8.2)

H **Home arrow**. Homes the cursor to the top line on the screen. If a count is given, then the cursor is moved to the count's line on the screen. In any case the cursor is moved to the first nonwhite character on the line. If used as the target of an operator, full lines

are affected.
(3.3, 4.2)

I Inserts at the beginning of a line; a synonym for **↑i**.

J Joins lines together, supplying appropriate white space: one space between words, two spaces after a ., and no spaces at all if the first character of the joined on line is). A count causes that many lines to be joined rather than the default two.
(7.5, 8.1f)

K Unused.

L Moves the cursor to the first nonwhite character of the last line on the screen. With a line count number, moves the cursor to the first nonwhite character of the indicated line from the bottom. Operators affect whole lines when used with **L**.
(3.3)

M Moves the cursor to the middle line on the screen, at the first nonwhite position on the line.
(3.3)

N Scans for the next match of the last pattern given to **/** or **?**, but in the reverse direction; this is the reverse of **n**.

O Opens a new line above the current line and inputs text there up to an ESC. A count can be used on dumb terminals to specify a number of lines to be opened; this is generally obsolete, as the *slowopen* option works better.
(4.1)

P Puts the last deleted text back before/above the cursor. The text goes back as whole lines above the cursor if it was deleted as whole lines; otherwise, the text is inserted between the characters before and at the cursor. The **P** character may be preceded by a named buffer specification "*x* to retrieve the contents of the buffer; buffers **1-9** contain deleted material, buffers **a-z** are available for general use.
(7.3)

Q Quits from **vi** to **ex** command mode. In this mode, whole lines form commands, ending with a RETURN. You can give all the : commands; the editor supplies the : as a prompt.
(8.7)

R Replaces characters on the screen with characters you type (overlay fashion). Terminates with an ESC.

S Changes whole lines; a synonym for **cc**. A count substitutes for that many lines. The lines are saved in the numeric buffers and erased on the screen before the substitution begins.

T Takes a single following character, locates the character before the cursor in the current line, and places the cursor just after that character. A count repeats the effect. Most useful with operators such as **d**.
(5.1)

U Restores the current line to its state before you started changing it.
(4.5)

V Unused.

W Moves forward to the beginning of a word in the current line, where words are defined as sequences of blank/nonblank characters. A count repeats the effect.
(3.4)

X Deletes the character before the cursor. A count repeats the effect, but only characters on the current line are deleted.

Y Yanks a copy of the current line into the unnamed buffer, to be put back by a later **p** or **P**; a synonym for **yy**. A count yanks that many lines. May be preceded by a buffer name to put lines in that buffer.
(8.4)

ZZ Exits the editor (same as **:x**CR). If any changes have been made, the buffer is written out to the current file. Then the editor quits.

[[Backs up the previous section boundary. A section begins at each macro in the *sections* options, normally a .NH or .SH and also at lines which start with a form feed **^L**. Lines beginning with { also stop **[[**; this makes it useful for looking backward, a function at a time, in C programs. If the option *lisp* is set, stops at each (at the beginning of a line, and is thus useful for moving backwards at the top level LISP objects.
(5.2, 7.1, 7.6, 8.2)

**** Unused.

]] Forward to a section boundary, see **[[** for a definition.
(5.2, 7.1, 7.6, 8.2)

↑ Moves to the first nonwhite position on the current line.
(5.4)

_ Unused

' When followed by a ' returns to the previous context. The previous context is set whenever the current line is moved in a nonrelative way. When followed by a letter **a-z**, returns to the position which was marked with this letter with an **m** command. When used with an operator such as **d**, the operation takes place from the exact marked place to the current position within the line; if you use **'**, the operation takes place over complete lines.
(3.2, 6.3)

a Appends arbitrary text after the current cursor position; the insert can continue onto multiple lines by using RETURN within the insert. A count causes the inserted text to be replicated, but only if the inserted text is all on one line. The insertion terminates with an ESC.
(4.1, 8.2)

b Backs up to the beginning of a word in the current line. A word is a sequence of alphanumerics, or a sequence of special characters. A count repeats the effect.
(3.4)

c An operator which changes the following object, replacing it with the following input text up to an ESC. If more than part of a single line is affected, the text which is changed away is saved in the numeric named buffers. If only part of the current line is affected, the last character to be changed away is marked with a **$**. A count causes that many objects to be affected, thus both **3c)** and **c3)** change the following three sentences.
(8.4)

d An operator which deletes the following object. If more than part of a line is affected, the text is saved in the numeric buffers. A count causes that many objects to be affected; thus **3dw** is the same as **d3w**.
(4.3, 4.4, 5.1, 8.4)

e Advances to the end of the next word, defined as for **b** and **w**. A count repeats the effect.
(3.4, 4.1)

f Finds the first instance of the next character following the cursor on the current line. A count repeats the find.
(5.1)

g Unused.

h **Left arrow**. Moves the cursor one character to the left. Like the other arrow keys, either **h**, the **left arrow** key, or one of the synonyms (**^H**) has the same effect. On V2 editors, arrow keys on certain kinds of terminals (those which send escape sequence, such as vt52, c100, or hp) cannot be used. A count repeats the effect.
(4.1, 8.5)

i Inserts text before the cursor, otherwise like **a**.
(8.2)

j **Down arrow**. Moves the cursor one line down in the same column. If the position does not exist, **vi** comes as close as possible to the same column. Synonyms include **^J** (line feed) and **^N**.

k **Up arrow**. Moves the cursor one line up. **^P** is a synonym.

l **Right arrow**. Moves the cursor one character to the right. SPACE is a synonym.

m Marks the current position of the cursor in the mark register which is specified by the next character **a-z**. Return to this position or use with an operator using **'** or **'**.
(6.3)

n Repeats the last **/** or **?** scanning commands.
(3.2)

o Opens new lines below the current line; otherwise like **O**.
(4.1)

p Puts text after/below the cursor; otherwise like **P**.
(7.3)

q Unused.

r Replaces the single character at the cursor with a single character you type. The new character may be a RETURN; this is the easiest way to split lines. A count replaces each of the following count characters with the single character given; see **R** above, which is usually the more useful iteration of **r**.
(4.2)

s Changes the single character under the cursor to the text which follows up to an ESC; given a count, that many characters from the current line are changed. The last character to be changed is marked with **$** as in **c**.
(4.2)

t Advances the cursor up to the character before the next character typed. Most useful with operator such as **d** and **c** to delete the characters up to a following character. You can use **.** to delete more if this does not delete enough the first time.
(5.1)

u Undoes the last change made to the current buffer. If repeated, will alternate between these two states, thus is its own inverse. When used after an insert which inserted text on more than one line, the lines are saved in the numeric named buffers.
(4.5)

v Unused.

w Advances to the beginning of the next word, as defined by **b**.
(3.4)

x Deletes the single character under the cursor. With a count, deletes that many characters forward from the cursor position, but only on the current line.
(7.5)

y An operator which yanks the following object into the unnamed temporary buffer. If preceded by a named buffer specification, "*x*, the text is placed in that buffer also. Text can be recovered by a later **p** or **P**.
(8.4)

z Redraws the screen with the current line placed as specified by the following character:

RETURN specifies the top of the screen
. specifies the center of the screen
- specifies the bottom of the screen.

A count may be given after the **z** and before the following character to specify the new screen size for the redraw. A count before the **z** gives the number of the line to place in the center of the screen instead of the default current line.
(5.4)

{ Retreats to the beginning of the preceding paragraph. A paragraph begins at each macro in the *paragraphs* option, normally .IP, .LP, .PP, .QP and **.bp**. A paragraph also begins after a completely empty line, and at each section boundary (see [[).
(5.2, 7.8, 8.6)

| Places the cursor on the character in the column specified by the count.
(8.1, 8.2)

} Advances to the beginning of the next paragraph. See { for the definition of paragraph.
(5.2, 7.8, 8.6)

~ Unused.

^? (DEL) Interrupts the editor, returning it to command-accepting state.
(2.4, 8.5)

APPENDIX E

Ex Reference Manual

Ex Reference Manual©

The ex editor† is a line oriented text editor, which supports both command and display oriented editing. This reference manual describes the command oriented part of ex.

© Copyright 1979, 1980, 1982 Regents of the University of California, reprinted by permission.

† The *vi*(*ex*) editor is based on software developed by The University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

1. Starting the ex Editor

When invoked, **ex** determines the terminal type from the TERM variable in the environment. If there is a TERMCAP variable in the environment and the type of the terminal described there matches the TERM variable, then that description is used. Also if the TERMCAP variable contains a pathname (beginning with a /) then the editor will seek the description of the terminal in that file (rather than the default /etc/termcap). If there is a variable EXINIT in the environment then the editor will execute the commands in that variable; otherwise, if there is a file *.exrc* in your HOME directory **ex** reads commands from that file, simulating a *source* command. Option setting commands placed in EXINIT or *.exrc* will be executed before each editor session.

A command to enter **ex** has the following prototype.[1]

ex [-][-v][-t *tag***][-r][-l][-w***n***][-x][-R][+***command***] name...**

- The most common case edits a single file with no options, i.e.:

  ex name

- The - command line option suppresses all interactive-user feedback and is useful in processing editor scripts in command files.
- The **-v** option is equivalent to using **vi** rather than **ex**.
- The **-t** option is equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.
- The **-r** option is used in recovering after an editor or system crash, retrieving the last saved version of the named file or, if no file is specified, typing a list of saved files.
- The **-l** option sets up for editing LISP, setting the *showmatch* and *lisp* options.
- The **-w** option sets the default window size to *n*, and is useful on dial-ups to start in small windows.
- The **-x** option causes **ex** to prompt for a *key*, which is used to encrypt and decrypt the contents of the file, which should already be encrypted using the same key, see *crypt*(1).
- The **-R** option sets the *readonly* option at the start.[2]
- The *name* arguments indicate files to be edited.
- An argument of the form +*command* indicates that the editor should begin by executing the specified command. If *command* is omitted, then it defaults to "$", positioning the editor at the last line of the first file initially. Other useful commands here are scanning patterns of the form "/pat" or line numbers, e.g., "+100" starting at line 100.

2. File Manipulation

2.1 Current File

The **ex** editor is normally editing the contents of a single file, whose name is recorded in the current file name. The **ex** editor performs all editing actions in a buffer (a temporary file) into which the text of the file is initially read. Changes made to the buffer have no effect on the file being edited until the buffer contents are written out to the file with a *write* command. After the buffer contents are written, the previous contents of the written file are no longer accessible. When a file is edited, its name becomes the current file name and its contents are read into the buffer.

---

1. Brackets ([]) surround optional parameters.
2. Not available in all V2 editors due to memory constraints.

The current file is almost always considered to be edited. This means that the contents of the buffer are logically connected with the current file name, so that writing the current buffer contents onto that file, even if it exists, is a reasonable action. If the current file is not edited then ex will not normally write on it if it already exists. The *file* command will say "[Not edited]" if the current file is not considered edited.

2.2 Alternate File

Each time a new value is given to the current file name, the previous current file name is saved as the alternate file name. Similarly, if a file is mentioned but does not become the current file, it is saved as the alternate file name.

2.3 Filename Expansion

Filenames within the editor may be specified using the normal shell expansion conventions. In addition, the character '%' in filenames is replaced by the current file name and the character '#' by the alternate file name. This makes it easy to deal alternately with two files and eliminates the need for retyping the name supplied on an **edit** command after a *No write since last change* diagnostic message is received.

2.4 Multiple Files and Named Buffers

If more than one file is given on the command line, then the first file is edited as described above. The remaining arguments are placed with the first file in the argument list. The current argument list may be displayed with the **args** command. The next file in the argument list may be edited with the **next** command. The argument list may also be respecified by specifying a list of names to the **next** command. These names are expanded with the resulting list of names becoming the new argument list, and **ex** edits the first file on the list.

For saving blocks of text while editing, and especially when editing more than one file, **ex** has a group of named buffers. These are similar to the normal buffer, except that only a limited number of operations are available on them. The buffers have names *a* through *z*. It is also possible to refer to A through Z; the upper case buffers are the same as the lower but commands **append** to named buffers rather than replacing if upper case names are used.

2.5 Read-Only Mode

It is possible to use **ex** in the read-only mode to look at files that you have no intention of modifying. This mode protects you from accidently overwriting the file. Read-only mode is on when the *readonly* option is set. It can be turned on with the **-R** command line option, by the *view* command line invocation, or be setting the *readonly* option. It can be cleared by setting the *noreadonly* mode. It is possible to write, even while in read-only mode, by indicating that you really know what you are doing. You can write to a different file, or use the **!** form of write, even while in read-only mode.

3. Exceptional Conditions

3.1 Errors and Interrupts

When errors occur **ex** (optionally) rings the terminal bell and prints an error diagnostic. If the primary input is from a file, editor processing will terminate. If an interrupt signal is received, ex prints "Interrupt" and returns to its command level. If the primary input is a file, then **ex** will exit when this occurs.

3.2 Recovering From Hang-ups and Crashes

If a hang-up signal is received and the buffer has been modified since it was last written, or if the system crashes, either the editor (in the first case) or the system (after it reboots in the second case) will attempt to preserve the buffer. The next time you log in you should be able to recover the work you were doing, losing at most a few lines of changes from the last point before the hang-up or editor crash. To recover a file you can use the **-r** option. If you were editing the file *resume*, then

you should change to the directory where you were when the crash occurred, giving the command

ex -r *resume*

After checking that the retrieved file is good, you can write it over the previous contents of that file.

You will normally get mail from the system telling you when a file has been saved after a crash. The command

ex -r

will print a list of the files which have been saved for you. In the case of a hang-up, the file will not appear in the list, although it can be recovered.

4. Editing Modes

The **ex** editor has five distinct modes.

- The primary mode is the command mode. Commands are entered in command mode when a : prompt is present and are executed each time a complete line is sent.
- In text input mode **ex** gathers input lines and places them in the file. The append, insert, and change commands use text input mode. No prompt is printed. This mode is left by typing a "." alone at the beginning of a line and **command** mode resumes.

The last three modes are *open* mode, *visual* mode (entered by the commands of the same name), and *text insertion* mode (within *open and visual* modes).

- The *open* mode allows local editing operations to be performed on the text in the file. The **open** command displays one line at a time and can be used on any terminal.
- The *visual* mode allows local editing operations to be performed on the text in the file. The **visual** command works on CRT terminals with random positioning cursors, using the screen as a single window for file editing changes.

These modes are described in sections 1 through 9.

5. Command Structure

Most command names are English words (initial prefixes of the words are acceptable abbreviations). The ambiguity of abbreviations is resolved in favor of the more commonly used commands. As an example, the **substitute** command can be abbreviated "s". The shortest available abbreviation for the **set** command is "se".

5.1 Command Parameters

Most commands accept prefix addresses specifying the lines in the file upon which they are to have effect. The forms of these addresses will be discussed below. A number of commands also may take a trailing count specifying the number of lines to be involved in the command.[3] Thus the **10p** command will print the tenth line in the buffer. The **delete 5** command will delete five lines from the buffer, starting with the current line.

Some commands take other information or parameters, the information always being given after the command name. Examples would be option names in a **set** command (**set** *number*), a file name in an **edit** command, a regular expression in a **substitute** command, or a target address for a **copy** command (1,5 copy 25).

---

3. Counts are rounded down if necessary.

5.2 Command Variants

A number of commands have two distinct variants. The variant form of the command is invoked by placing an ! immediately after the command name. Some of the default variants may be controlled by options; in this case the ! serves to toggle the default.

5.3 Flags After Commands

The characters #, **p** and **l** may be placed after many commands.[4] In this case, the command abbreviated by these characters is executed after the command completes. Since **ex** normally prints the new current line after each change, **p** is rarely necessary. Any number of + or - characters may also be given with these flags. If they appear, the specified offset is applied to the current line value before the printing command is executed.

5.4 Comments

It is possible to give editor commands which are ignored. This is useful when making complex editor scripts for which comments are desired. The comment character is the double quote ". Any command line beginning with " is ignored. Comments beginning with " may also be placed at the ends of commands, except in cases where they could be confused as part of the text (shell escapes and the substitute and map commands).

5.5 Multiple Commands Per Line

More than one command may be placed on a line by separating each pair of commands by a | character. However, the global commands, comments, and the shell escape ! must be the last command on a line, as they are not terminated by a |.

5.6 Reporting Large Changes

Most commands which change the contents of the editor buffer give feedback if the scope of the change exceeds a threshold given by the *report* option. This feedback helps to detect undesirably large changes so that they may be quickly and easily reversed with an **undo** command. After commands with more global effect (such as **global** or **visual**), you will be informed if the net change in the number of lines in the buffer during this command exceeds this threshold.

6. Command Addressing

6.1 Addressing Primitives

The current line.	Most commands leave the current line as the last line which they affect. The default address for most commands is the current line, thus . is rarely used alone as an address.
n	The *n*th line in the editor buffer, lines being numbered sequentially from 1.
$	The last line in the buffer.
%	An abbreviation for "1,$", the entire buffer.
+n -n	An offset relative to the current buffer line. The forms .+3, +3, and +++ are all equivalent; if the current line is line 100 they all address line 103.
/pat/ ?pat?	Scan forward and backward respectively for a line containing *pat*, a regular expression (as defined below). The scans normally wrap around the end of the buffer. If all that is desired is to print the next line containing *pat*, then the trailing / or ? may be omitted. If *pat* is omitted or explicitly empty,

4. A **p** or **l** must be preceded by a blank or tab except in the single special case **dp**.

then the last regular expression specified is located.[5]

" 'x — Before each non-relative motion of the current line ., the previous current line is marked with a tag, subsequently referred to as ". This makes it easy to refer or return to this previous context. Marks may also be established by the *mark* command, using single lower case letters *x* and the marked lines referred to as '*x*.

6.2 Combining Addressing Primitives

Addresses to commands consist of a series of addressing primitives, separated by , or ;. Such address lists are evaluated left-to-right. When addresses are separated by , the current line . is set to the value of the previous addressing expression before the next address is interpreted. If more addresses are given than the command requires, then all but the last one or two are ignored. If the command takes two addresses, the first addressed line must precede the second in the buffer. Null address specifications are permitted in a list of addresses, the default in this case is the current line (.); thus, **,100** is equivalent to **.,100**. It is an error to give a prefix address to a command which expects none.

7. Command Descriptions

The following form is a prototype for all **ex** commands:

address **command** ! *parameters count flags*

All parts are optional; the degenerate case is the empty command which prints the next line in the file. For sanity with use from within *visual* mode, the **ex** editor ignores a . preceding any command. In the following command descriptions, the default addresses are shown in parentheses.

Note: The parentheses are not part of the command syntax.

abbreviate *word rhs* — abbr: **ab**

Add the named abbreviation to the current list. When in the input mode in **visual**, if *word* is typed as a complete word, it will be changed to *rhs*.

(.) **append** — abbr: **a**
text
.

Reads the input text and places it after the specified line. After the command, . addresses the last line input or the specified line if no lines were input. If address **0** is given, text is placed at the beginning of the buffer.

a!
text
.

The variant flag to *append* toggles the setting for the *autoindent* option during the input of *text*.

args

The members of the argument list are printed, with the current argument delimited by [and].

(.,.) **change** *count* — abbr: **c**

---

5. The forms \/ and \? scan using the last regular expression used in a scan; after a substitute, // and ?? would scan using the substitute's regular expression.

text

.

Replaces the specified lines with the input *text*. The current line becomes the last line input; if no lines were input it is left as for a **delete**.

c!
text

.

The variant toggles *autoindent* during the change.

(.,.) **copy** *addr flags* abbr: **co**

A copy of the specified lines is placed after *addr*, which may be **0**. The current line . addresses the last line of the copy. The command **t** is a synonym for **copy**.

(.,.) **delete** *buffer count flags* abbr: **d**

Removes the specified lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line. If a named *buffer* is specified by giving a letter, the specified lines are saved in that buffer or appended to it if an upper case letter is used.

ex *file* abbr: **e**

Used to begin an editing session on a new file. The editor first checks to see if the current buffer has been modified since the last **write** command was issued. If it has been, a warning is issued and the command is aborted. The command otherwise deletes the entire contents of the editor buffer, makes the named file the current file, and prints the new filename. After insuring that this file is not a binary file (such as a directory), a block or character special file (other than */dev/tty)*, a terminal, or a binary or executable file (as indicated by the first word), the editor reads the file into its buffer. If the read of the file completes without error, the number of lines and characters read is typed. If there were any non-ASCII characters in the file they are stripped of their non-ASCII high bits, and any null characters in the file are discarded. If none of these errors occurred, the file is considered edited. If the trailing newline character is missing from the last line of the input file, it will be supplied and a complaint will be issued. This command leaves the current line (.) at the last line read. If executed from within *open* or *visual* mode, the current line is initially the first line of the file.

e! *file*

The variant form suppresses the complaint about modifications having been made and not written from the editor buffer, thus discarding all changes which have been made before editing the new file.

e +n *file*

Causes the editor to begin at line *n* rather than at the last line; *n* may also be an editor command containing no spaces, e.g., **+/pat**.

file abbr: **f**

Prints:

the current file name

whether it has been "modified" since the last **write** command

whether it is *read-only* mode

the current line

the number of lines in the buffer

the percentage of the way through the buffer of the current line.

In the rare case that the current file is "not edited" this is noted also. In this case you have to use the form **w!** to write to the file, since the editor is not sure that a **write** command will not destroy a file unrelated to the current contents of the buffer.

file *file*
The current file name is changed to *file* which is considered "not edited".

(1,$) **global** */pat/cmds* abbr: **g**
First marks each line among those specified which matches the given regular expression. Then the given command list is executed with . initially set to each marked line.

The command list consists of the remaining commands on the current input line and may continue to multiple lines by ending all but the last such line with a \. If *cmds* (and possibly the trailing / delimiter) is omitted, each line matching *pat* is printed. The **append**, **insert**, and **change** commands and associated input are permitted; the . terminating input may be omitted if it would be on the last line of the command list. The **open** and **visual** commands are permitted in the command list and take input from the terminal.

The **global** command itself may not appear in *cmds*. The **undo** command is also not permitted there, since **undo** instead can be used to reverse the entire **global** command. The options *autoprint* and *autoindent* are inhibited during a **global** command, (and possibly the trailing / delimiter) and the value of the *report* option is temporarily infinite, in deference to a *report* for the entire **global** command. Finally, the context mark (") is set to the value of . before the **global** commands begin and is not changed during a **global** command, except perhaps by an *open* **or** *visual* mode within the **global** command.

g! */pat/cmds* abbr: **v**
The variant form of a **global** command runs *cmds* at each line not matching *pat*.

(.) **insert** abbr: **i**
text
.

Places the given text before the specified line. The current line is left at the last line input. If there were no lines input it is left at the line before the addressed line. This command differs from **append** only in the placement of text.

i!
text
.

The variant toggles *autoindent* during the insert.

(.,.+1) **join** *count flags* abbr: **j**

Places the text from a specified range of lines together on one line. White space is adjusted at each junction to provide at least one blank character, two if there was a . at the end of the line, or none if the first following character is a). If there is already white space at the end of the line, the white space at the start of the next line will be discarded.

j!

The variant causes a simpler *join* with no white space processing. Characters in the lines are simply concatenated.

(.) **k** *x*

The **k** command is a synonym for **mark**. It does not require a blank or tab before the following letter.

(.,.) **list** *count flags*

Prints the specified lines in a more unambiguous way. Tabs are printed as ^I and the end of each line is marked with a trailing **$**. The current line is left at the last line printed.

map *lhs rhs*

The **map** command is used to define macros for use in *visual* mode. The *lhs* should be a single character, or the sequence *#n* (for a digit), referring to function key *n*. When this character or function key is typed in *visual* mode, it will be as though the corresponding *rhs* has been typed. On terminals without function keys, you can type *#n*. See section 7.9 for more details.

(.) **mark** *x*

Gives the specified line mark *x*, a single lower case letter. The *x* must be preceded by a blank or a tab. The addressing form **'x** then addresses this line. The current line is not affected by this command.

(.,.) **move** *addr* abbr: **m**

The **move** command repositions the specified lines to be after *addr*. The first of the moved lines becomes the current line.

next abbr: **n**

The next file from the command line argument list is edited.

n!

The variant suppresses warnings about the modifications to the buffer not having been written out, discarding (irretrievably) any changes which may have been made.

n *filelist*
n *+command filelist*

The specified *filelist* is expanded and the resulting list replaces the current argument list. The first file in the new list is then edited. If *command* is given (it must contain no spaces), then it is executed after editing the first such file.

(.,.) **number** *count flags* abbr: **#** or **nu**

Prints each specified line preceded by its buffer line number. The current line is left at the last line printed.

(.) **open** *flags* abbr: **o**
(.) **open** */patflags*

Enters intraline editing *open* mode at each addressed line. If *pat* is given, then the cursor will be placed initially at the beginning of the string matched by the pattern. To exit this mode

use Q. See sections 1 through 9 for more details.[6]

preserve

The current editor buffer is saved as though the system has just crashed. This command is for use only in emergencies when a **write** command has resulted in an error and you do not know how to save your work. After a **preserve** you should seek help.

(.,.)**print** *count* abbr: **p** or **P**

Prints the specified lines with non-printing characters printed as control characters ^*x*; delete (octal 177) is represented as ^?. The current line is left at the last line printed.

(.)**put** *buffer* abbr: **pu**

Puts back previously deleted or yanked lines. Normally used with **delete** to effect movement of lines, or with **yank** to effect duplication of lines. If no buffer is specified, then the last deleted or yanked text is restored.[7] By using a named buffer, text may be restored that was saved there at any previous time.

quit abbr: **q**

Causes the **ex** editor to terminate. No automatic write of the editor buffer to a file is performed. However, **ex** issues a warning message if the file has changed since the last **write** command was issued, and does not **quit** (the **ex** editor will also issue a diagnostic if there are more files in the argument list). Normally, you will wish to save your changes and you should give a **write** command. If you wish to discard them, use the **q!** command variant.

q!

Quits from the editor, discarding changes to the buffer without complaint.

(.)**read** *file* abbr: **r**

Places a copy of the text of the given file in the editing buffer after the specified line. If no *file* name is given, the current file name is used. The current file name is not changed unless there is none, in which case *file* becomes the current name. The sensibility restrictions of the **edit** command apply here also. If the file buffer is empty and there is no current name, then **ex** treats this as an **edit** command.

Address **0** is legal for this command and causes the file to be read at the beginning of the buffer. Statistics are given as for the **edit** command when the **read** successfully terminates. After a **read** the current line is the last line read.[8]

(.)**read !***command*

Reads the output of the command **command** into the buffer after the specified line. This is not a variant form of the command, rather a read specifying a *command* rather than a *filename*. A blank or tab before the **!** is mandatory.

recover *file*

Recovers *file* from the system save area. Used after an accidental hang-up of the phone or a system crash[9] or **preserve** command, except that when you use **preserve** you will be notified by

---

6. Not available in all V2 editors due to memory constraints.
7. No modifying commands may intervene between the **delete** or **yank** and the **put**, nor may lines be moved between files without using a named buffer.
8. Within *open* and *visual* the current line is set to the first line read rather than the last.
9. The system saves a copy of the file you are editing only if you have made changes to the file.

mail when a file is saved.

rewind abbr: **rew**

The argument list is rewound, and the first file in the list is edited.

rew!

Rewinds the argument list discarding any changes made to the current buffer.

set *parameter*

With no arguments, prints those options whose values have been changed from their defaults; with parameter *all* it prints all of the option values.

Giving an option name followed by a **?** causes the current value of that option to be printed. The **?** is unnecessary unless the option is Boolean valued. Boolean options are given values either by the form "set *option*" to turn them on or "set *nooption*" to turn them off. String and numeric options are assigned via the form "set *option*=value".

More than one parameter may be given to **set**; they are interpreted left-to-right.

shell abbr: **sh**

A new shell is created. When it terminates, editing resumes.

source *file* abbr: **so**

Reads and executes commands from the specified file. The **source** commands may be nested.

(.,.)**substitute** */pat/repl/options count flags* abbr: **s**

On each specified line, the first instance of pattern *pat* is replaced by replacement pattern *repl*. If the *global* indicator option character **g** appears, then all instances are substituted. If the *confirm* indication character **c** appears, then before each substitution, the line to be substituted is typed with the string to be substituted marked with ↑ characters. By typing a **y**, one can cause the substitution to be performed; any other input causes no change to take place. After a **substitute** command the current line is the last line substituted.

Lines may be split by substituting newline characters into them. The newline in *repl* must be escaped by preceding it with a \. Other metacharacters available in *pat* and *repl* are described below.

(.,.) **substitute** *options count flags* abbr: **s**

If *pat* and *repl* are omitted, then the last substitution is repeated. This is a synonym for the **&** command.

(.,.) **t** *addr flags*

The **t** command is a synonym for *copy*.

ta *tag*

The focus of editing switches to the location of *tag*, switching to a different line in the current file where it is defined, or if necessary to another file. If you have modified the current file before giving a *tag* command, you must write it out; giving another *tag* command, specifying no *tag* will reuse the previous tag.

The *tag* file is normally created by a program such as **ctags**, and consists of a number of lines with three fields separated by blanks or tabs. The first field gives the name of the tag, the second the name of the file where the tag resides, and the third gives an addressing form which can be used by the editor to find the tab. This field is usually a contextual scan using

/pat/ to be immune to minor changes in the file. Such scans are always performed as if *nomagic* was set.

Names in the *tag* file must be sorted alphabetically.[10]

unabbreviate *word* abbr: **una**
Delete *word* from the list of abbreviations.

undo abbr: **u**
Reverses the changes made in the buffer by the last buffer editing command.

Note: **global** commands are considered a single command for the purpose of **undo** (as are **open** and **visual**). Also, the commands **write** and **edit** which interact with the file system cannot be undone; **undo** is its own inverse.

The **undo** command always marks the previous value of the current line (.) as ". After an **undo** command, the current line is the first line restored or the line before the first line deleted if no lines were restored. For commands with more global effect such as **global** and **visual** the current line regains it's pre-command value after an **undo**.

unmap *lhs*
The macro expansion associated by **map** for *lhs* is removed.

(1,$) *v/pat/cmds*
A synonym for the **global** command variant **g!**, running the specified *cmds* on each line which does not match *pat*.

version abbr: **ve**
Prints the current version number of the editor as well as the date the editor was last changed.

(.) **visual** *type count flags* abbr: **vi**
Enters visual mode at the specified line. The *type* argument is optional and may be -, ↑, or . as in the **z** command to specify the placement of the specified line on the screen. By default, if *type* is omitted, the specified line is placed as the first on the screen. A *count* specifies an initial window size; the default is the value of the option *window*. See sections 1 through 9 for more details. To exit this mode, type **Q**.

visual file
visual +*n* file
From visual mode, this command is the same as edit

(1,$) **write** *file* abbr: **w**
Write changes made back to *file*, printing the number of lines and characters written. Normally *file* is omitted and the text goes back where it came from. If a *file* is specified, then text will be written to that file.[11] If the file does not exist it is created. The current file name is changed only if there is no current file name; the current line is never changed.

---

10. Not available in all V2 editors due to memory constraints.

11. The editor writes to a file only if it is the current file and is *edited*, if the file does not exist, or if the file is actually a teletype (*/dev/tty*, */dev/null*). Otherwise, you must give the variant form **w!** to force the write.

If an error occurs while writing the current and *edited* file, the editor considers that there has been no write since last change even if the buffer had not previously been modified.

(1,$) **write>>***file* abbr: **w>>**
Writes the buffer contents at the end of an existing file.

w! *name*
Overrides the checking of the normal **write** command and will write to any file which the system permits.

(1,$) **w !***command*
Writes the specified lines into *command*.

Note: There is a difference between **w!** which overrides checks and **w !** which writes to a command.

wq *name*
Like a write and then a **quit** command.

wq! *name*
The variant overrides checking on the sensibility of the **write** command, as **w!** does.

xit *name*
If any changes have been made and not written, writes the buffer out. Then, in any case, quits.

(.,.)**yank** *buffer count* abbr: **ya**
Places the specified lines in the named buffer, for later retrieval via **put**. If no buffer name is specified, the lines go to a more volatile place (see the **put** command description).

(.+1) **z** *count*
Print the next *count* lines (default *window*).

(.) **z** *type count*
Prints a window of text with the specified line at the top. If *type* is - the line is placed at the bottom, a . causes the line to be placed in the center.

Note: Forms **z=** and **z↑** also exist; **z=** places the current line in the center, surrounds it with lines of - characters, and leaves the current line at this line. The form **z↑** prints the window before **z-** would. The characters **+**, **↑** and **-** may be repeated for cumulative effect. On some V2 editors, no *type* may be given.

A count gives the number of lines to be displayed rather than double the number specified by the *scroll* option. On a CRT the screen is cleared before display begins unless a count which is less than the screen size is given. The current line is left at the last line printed.

! *command*
The remainder of the line after the ! character is sent to a shell to be executed. Within the text of *command* the characters % and # are expanded as in file names, and the ! character is replaced with the text of the previous command. Thus, in particular, !! repeats the last such shell escape. If any such expansion is performed, the expanded line will be echoed. The current line is unchanged by this command.

If there has been "no write" of the buffer contents since the last change to the editing buffer,

then as a warning, a diagnostic message will be printed before the command is executed. A single ! is printed when the command completes.

(addr, addr) ! command
Takes the specified address range and supplies it as standard input to *command*. The resulting output then replaces the input lines.

($) =
Prints the line number of the addressed line. The current line is unchanged.

(.,.) > *count flags*
(.,.) < *count flags*
Perform intelligent shifting on the specified lines: < shifts left and > shifts right. The quantity of shift is determined by the *shiftwidth* option and the repetition of the specification character. Only white space (blanks and tabs) is shifted. No non-white characters are discarded in a left-shift. The current line becomes the last line which changed due to the shifting.

^D
An end-of-file from a terminal input scrolls through the file. The *scroll* option specifies the size of the scroll, normally a half screen of text.

(.+1,.+1)
(.+1,.+1)|
An address alone causes the addressed lines to be printed. A blank line prints the next line in the file.

(.,.) & *options count flags*
Repeats the previous substitute command.

(.,.) ~ *options count flags*
Replaces the previous regular expression with the previous replacement pattern from a substitution.

8. Regular Expressions and Substitute Replacement Patterns

8.1 Regular Expressions

A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression. The **ex** editor remembers two previous regular expressions: the previous regular expression used in a substitute command and the previous regular expression used elsewhere (referred to as the previous *scanning* regular expression). The previous regular expression can always be referred to by a null *re* (// or ??).

8.2 Magic and Nomagic

The regular expressions allowed by the **ex** editor are constructed in one of two ways depending on the setting of the *magic* option. The **ex** and **vi** editor default setting of *magic* gives quick access to a powerful set of regular expression metacharacters. The disadvantage of *magic* is that the user must remember that these metacharacters are *magic* and precede them with the character \ to use them as "ordinary" characters. With *nomagic*, regular expressions are much simpler, there being only two metacharacters. The power of the other metacharacters is still available by preceding the (now) ordinary character with a \.

Note: \ is always a metacharacter.

The remainder of the discussion of regular expressions assumes that the setting of this option is *magic*.[12]

8.3 Basic regular expression summary

The following basic constructs are used to construct *magic* mode regular expressions.

char An ordinary character matches itself. The characters ↑ at the beginning of a line, **$** at the end of line, * as any character other than the first, ., \, [, and ~ are not ordinary characters and must be escaped (preceded) by \ to be treated as such.

↑ At the beginning of a pattern forces the match to succeed only at the beginning of a line.

$ At the end of a regular expression forces the match to succeed only at the end of the line.

. Matches any single character except the newline character.

\< Forces a match to occur only at the beginning of a variable or word; i.e., either at the beginning of a line, or just before a letter, digit, or underline and after a character which is not one of these.

\> Forces a match to occur only at the end of a variable or word, i.e., either the end of the line or before a character which is neither a letter, a digit, nor the underline character.

[*string*] Matches any single character in the class defined by *string*. Most characters in *string* define themselves. A pair of characters separated by - in *string* defines the set of characters collating between the specified lower and upper bounds, thus **[a-z]** as a regular expression matches any single lower-case letter. If the first character of *string* is an ↑ then the construct matches those characters which it otherwise would not; thus **[↑a-z]** matches anything but a lower-case letter (and of course a newline character). To place any of the characters ↑, [, or - in *string* you must escape them with a preceding \.

8.4 Combining Regular Expression Primitives

The concatenation of two regular expressions matches the leftmost and then longest string which can be divided, with the first piece matching the first regular expression and the second piece matching the second. Any of the (single-character matching) regular expressions mentioned above may be followed by the character * to form a regular expression which matches any number of adjacent occurrences (including 0) of characters matched by the regular expression it follows.

The character ~ may be used in a regular expression and matches the text which defined the replacement part of the last substitute command. A regular expression may be enclosed between the sequences \(and \) with side effects in the substitute replacement patterns.

8.5 Substitute Replacement Patterns

The basic metacharacters for the replacement pattern are **&** and ~; these are given as \& and \~ when *nomagic* is set. Each instance of **&** is replaced by the characters which the regular expression matched. The metacharacter ~ stands (in the replacement pattern) for the defining text of the previous replacement pattern.

Other metasequences possible in the replacement pattern are always introduced by the escape character \. The sequence *n* is replaced by the text matched by the *n*th regular subexpression

12. To discern what is true with *nomagic* it suffices to remember that the only special characters in this case will be ↑ at the beginning of a regular expression, **$** at the end of a regular expression, and \. With *nomagic* the characters ~ and **&** also lose their special meanings to the replacement pattern of a substitute.

enclosed between \(and \).[13] The sequences **\u** and **\l** cause the immediately following character in the replacement to be converted to upper case or lower case, respectively, if this character is a letter. The sequences **\U** and **\L** turn such conversion on, either until **\E** or **\e** is encountered or until the end of the replacement pattern.

9. Option Descriptions

autoindent, ai default: noai

Can be used to ease the preparation of structured program text. At the beginning of each **append**, **change**, or **insert** command or when a new line is opened or created by an *append*, *change*, *insert*, or *substitute* operation within the *open* or *visual* mode, **ex** looks at the line being appended after, the first line changed, or the line inserted before and calculates the amount of white space at the start of the line. It then aligns the cursor at the level of indentation so determined.

If the user then types in lines of text, the lines will continue to be justified at the displayed indenting level. If more white space is typed at the beginning of a line, the following line will start aligned with the first non-white character of the previous line. To back the cursor to the preceding tabstop one can hit **^D**. The tabstops (going backwards) are defined as multiples of the *shiftwidth* option. You cannot backspace over the indent, except by sending an end-of-file with a **^D**.

Specially processed in this mode is a line with no character added to it, which turns into a completely blank line (the white space provided for the *autoindent* is discarded). Also specially processed in this mode are lines beginning with an **↑** and immediately followed by a **^D**. This causes the input to be repositioned at the beginning of the line while retaining the previous indent for the next line. Similarly, a **0** followed by a **^D** repositions at the beginning without retaining the previous indent.

The *autoindent* option does not happen in **global** commands or when the input is not a terminal.

autoprint,ap default: ap

Causes the current line to be printed after each **delete**, **copy**, **join**, **move**, **substitute**, **t**, **undo** or **shift** command. This has the same effect as supplying a trailing **p** to each such command. The *autoprint* is suppressed in globals, and only applies to the last of many commands on a line.

autowrite,aw default: noaw

Causes the contents of the buffer to be written to the current file if you have modified it and gives a **next**, **rewind**, **tab**, or **!** command, or a **^↑** (switch files) or **^]** (tag goto) command in **visual**.

Note: The command does not autowrite. In each case, there is an equivalent way of switching when the *autowrite* option is set to avoid the autowrite (**ex** for **next**, **rewind!** for **rewind**, **tag!** for **tag**, **shell** for **!**, and **:e #** and a **:ta!** command from within **visual**).

beautify, bf default: nobeautify

Causes all control characters except tab, newline, and form-feed to be discarded from the

13. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left.

input. A complaint is registered the first time a backspace character is discarded. The *beautify* option does not apply to command input.

directory, dir default: dir=/tmp
Specifies the directory in which **ex** places its buffer file. If this directory in not writable, then the editor will exit abruptly when it fails to be able to create its buffer there.

edcompatible default: noedcompatible
Causes the presence or absence of **g** and **c** suffixes on substitute commands to be remembered and to be toggled by repeating the suffices. The suffix **r** makes the substitution be as in the ~ command, instead of like **&**.[14]

errorbells,eb default: noeb
Error messages are preceded by a bell.[15] If possible the editor always places the error message in a standout mode of the terminal (such as inverse video) instead of ringing the bell.

hardtabs, ht default: ht=8
Gives the boundaries on which terminal hardware tabs are set (or on which the system expands tabs).

ignorecase,ic default: noic
All upper case characters in the text are mapped to lower case in regular expression matching. In addition, all upper case characters in regular expressions are mapped to lower case except in character class specifications.

lisp default: nolisp
The *autoindent* option indents appropriately for *lisp* code, and the **()**, **{}**, **[[**, and **]]** commands in *open* and *visual* modes are modified to have meaning for *lisp*.

list default: nolist
All printed lines will be displayed more unambiguously, showing tabs and end-of-lines as in the **list** command.

magic default: magic for **ex** and **vi**
If *nomagic* is set, the number of regular expression metacharacters is greatly reduced, with only **↑** and **$** having special effects. In addition the metacharacters **~** and **&** of the replacement pattern are treated as normal characters. All the normal metacharacters may be made *magic* when *nomagic* is set by preceding them with a \.

mesg default: mesg
Causes write permission to be turned off to the terminal while you are in visual mode, if *nomesg* is set.[16]

number,nu default: nonumber
Causes all output lines to be printed with line numbers. In addition each input line will be prompted for by supplying the line number it will have.

---

14. Version 3 only.

15. Bell ringing in *open* and *visual* mode on errors is not suppressed by setting *noeb*.

16. Version 3 only.

open default: open

If *noopen*, the commands **open** and **visual** are not permitted.

optimize, opt default: optimize

Throughput of text is expedited by setting the terminal not to do automatic carriage returns when printing more than one (logical) line of output, greatly speeding output on terminals without addressable cursors when text with leading white space is printed.

paragraphs, para default: para=IPLPPPQPP LIbp

Specifies the paragraphs for the { and } operations in *open* and *visual*. The pairs of characters in the option's value are the names of the macros which start paragraphs.

prompt default: prompt

Command mode input is prompted for with a colon (:).

redraw default: noredraw

The editor simulates (using great amounts of output) an intelligent terminal on a dumb terminal (e.g., during insertions in *visual* the characters to the right of the cursor position are refreshed as each input character is typed). This option is useful only at very high speed.

remap default: remap

If on, macros are repeatedly tried until they are unchanged.[17] For example, if **o** is mapped to **O**, and **O** is mapped to **I**, then if *remap* is set, **o** will map to **I**, but if *noremap* is set, it will map to **O**.

report default: report=5

Specifies a threshold for feedback from commands. Any command which modifies more than the specified number of lines will provide feedback as to the scope of its changes. For commands such as **global**, **open**, **undo**, and **visual**, which have potentially more far-reaching scope, the net change in the number of lines in the buffer is presented at the end of the command, subject to this same threshold. Thus, notification is suppressed during a **global** command on the individual commands performed.

scroll default: scroll=½ window

Determines the number of logical lines scrolled when an end-of-file is received from a terminal input in command mode and the number of lines printed by a command mode **z** command (double the value of *scroll*).

sections default: sections=SHNHH HU

Specifies the section macros for the [[and]] operations in *open* and *visual*. The pairs of characters in the option's value are the names of the macros which start paragraphs.

shell, sh default: sh=/bin/sh

Gives the pathname of the shell forked for the shell escape command !, and by the **shell** command. The default is taken from SHELL in the environment, if present.

shiftwidth, sw default: sw=8

Gives the width a software tabstop used in reverse tabbing with ^**D** when using *autoindent* to append text, and by the shift commands.

---

17. Version 3 only.

showmatch, sm default: nosm

In *open* and *visual* mode, when a) or } is typed, moves the cursor to the matching (or { for one second if this matching character is on the screen. Extremely useful with *lisp*.

slowopen, slow terminal dependent

Affects the display algorithm used in *visual* mode, holding off display updating during input of new text to improve throughput when the terminal in use is both slow and unintelligent.

tabstop, ts default: ts=8

The editor expands tabs in the input file to be on *tabstop* boundaries for the purposes of display.

taglength, tl default: tl=0

Tags are not significant beyond this many characters. A value of zero (the default) means that all characters are significant.

tags default: tags=tags/usr/lib/tags

A path of files to be used as tag files for the *tag* command.[18] A requested tag is searched for in the specified files, sequentially. By default, files called **tags** are searched for in the current directory and in */usr/lib* (a master file for the entire system).

term from environment TERM

The terminal type of the output device.

terse default: noterse

Shorter error diagnostics are produced for the experienced user.

warn default: warn

Warn if there has been "[No write since last change]" before a ! command escape.

window default: window=speed dependent

The number of lines in a text window in the **visual** command. The default is 8 at slow speeds (600 baud or less), 16 at medium speed (1200 baud), and the full screen (minus one line) at higher speeds.

w300, w1200, w9600

These are not true options but set **window** only if the speed is slow (300), medium (1200), or high (9600), respectively. They are suitable for an EXINIT and make it easy to change the 8/16/full screen rule.

wrapscan, ws default: ws

Searches that use regular expressions in addressing will wrap around past the end of the file.

wrapmargin, wm default: wm=0

Defines a margin for automatic wrapover of text during input in *open* and *visual* modes.

writeany, wa default: nowa

Inhibit checks normally made before **write** commands, allowing a write to any file which the system protection mechanism will allow.

---

18. Version 3 only.

10. Limitations

Editor limits that the user is likely to encounter are as follows: 1024 characters per line, 256 characters per global command list, 128 characters per file name, 128 characters in the previous inserted and deleted text in *open* or *visual*, 100 characters in a shell escape command, 63 characters in a string valued option, and 30 characters in a tag name, and a limit of 250,000 lines if the file is silently enforced.

The *visual* implementation limits to 32 the number of macros defined with map, and the total number of characters in macros to be less than 512.

APPENDIX F

Uucp Tutorial

CONTENTS

1. Introduction 1

2. The Uucp Network 1
   2.1 Network Hardware 1
   2.2 Network Topology 2
   2.3 Forwarding 2
   2.4 Security 3
   2.5 Software Structure 3
   2.6 Rules of the Road 3
   2.7 Special Places: The Public Area 4
   2.8 Permissions 4

3. Network Usage 4
   3.1 Name Space 4
   3.2 Forwarding Syntax 5
   3.3 Types of Transfers 6
   3.4 Remote Executions 6
   3.5 Spooling 7
   3.6 Notification 7
   3.7 Tracking and Status 7
   3.8 Job Status 8
   3.9 Network Status 8
   3.10 Job Control 8

4. Utilities That Use Uucp 9
   4.1 Mail 9
   4.2 Netnews 9
   4.3 Other Applications 9

5. Conclusion 9

Uucp Tutorial

1. Introduction

The *uucp* network has provided a means of information exchange between UNIX[1] systems over the DDD network for several years [1]. This document is an attempt to provide the novice user with the background to make use of the network.

While it is perfectly reasonable for a user to assume that knowledge of the mechanism for transmission of information between UNIX Systems is unnecessary, in practice, it is often useful to understand basic principles so that the best possible use of the network can be made. Two earlier documents describe the goals and the details of an early implementation of *uucp [1, 2]*. This document covers much of the same ground but reflects the improvements that have been made to the system and the information is tailored to the novice user.

The first half of the document discusses concepts. The second half explains the use of the user level interface to the network and provides numerous examples.

2. The Uucp Network

The *uucp(1)* network is a network of UNIX systems that allows file transfer and remote execution to occur on a network of UNIX Systems. The *extent* of the network is a function of both the interconnection hardware and the controlling network software. Membership in the network is tightly controlled via the software to preserve the integrity of all members of the network. The following sections describe the topology, services, operating rules, etc. of the network in detail to provide a framework for discussing use of the network.

2.1 Network Hardware

Uucp(1) was originally designed as a dialup network so that systems in the network could use the DDD network to communicate with each other. The three most common methods of connecting systems are,

1. Two UNIX Systems can be directly connected by cross-coupling (via a null modem) two of the computers ports. This means of connection is useful for only short distances (several hundred feet can be achieved although the RS232 standard specifies a much shorter distance) and is usually run at high speed (9600 baud). These connections run on asynchronous terminal ports.
2. A modem (using a private line or a limited distance modem) can be used to *directly* connect processors over a private line (using 103 or 212 type datasets).
3. Lastly, and more commonly, a processor can use a modem and an automatic calling unit (ACU) to use the DDD network to contact another system. This is by far the most common interconnection method and makes available the largest number of connections.

In principle *uucp* could be extended to use higher speed media (e.g., HYPERchannel[2], Ethernet[3], etc.) and this possibility is being explored for future UNIX releases. Some sites already support local modifications to *uucp* for using Datakit, X.25 (permanent virtual circuits) and calling through dataswitches.

---

1. UNIX is a Trademark of Bell Telephone Laboratories, Incorporated.
2. HYPERchannel is a Trademark of Network Systems Corporation.
3. Ethernet is a Trademark of Xerox Corp.

2.2 Network Topology

A large number of connections between systems are possible via the DDD network. The topology of the network is, however, determined by both the hardware connections and the software that controls the network. The next two sections deal with how that topology is controlled.

2.2.1 Hardware Topology As discussed earlier, it is possible to build a network using permanent or dial up connections. In Figure 1, a group of systems (A, B, C, D and E) are shown as connected via hardwired lines (all systems are assumed to have some answer only datasets so remote users or systems can be connected). A few systems have automatic calling units (K, D, F and G) and one system (H) does not have any capability for calling other systems. Users should be aware that the network consists of a series of point to point connections (A-B, B-C, D-B, E-B) even though it appears in Figure 1 that A and C are directly connected through B. The following observations can be made,

1. System H is isolated. It can be made part of the network by arranging for other systems to *poll* it at fixed intervals. This is an important concept to remember since transfers from systems that are *polled* will not leave the system until that system is called by a polling system.
2. Systems K, F, G and D can easily reach all other systems since they have calling units.
3. If system A (E or G) wishes to send a file to H (K, F or G) it must first send it to D (via system B) since D is the only system with a calling unit.

2.2.2 Software Topology The hardware capability of systems in the network defines the *maximum* number of connections in the network. The software at each node restricts the access by other systems and thereby defines the extent of the network. The systems of Figure 1 can be configured so that conceptually they appear as a network of systems that have equal access to each other or some restrictions can be applied. As part of the security mechanism used by *uucp(1)*, the extent of access that other systems have can be controlled at each node. Figure 2a,b shows how the network might appear at one node. Access is available from all systems in Figure 2a, however, in Figure 2b some of the systems have been configured to have greater or less access privileges than others (i.e., system C, E, G have one set of access privileges, systems F and B have another set, etc.).

Uucp uses the UNIX password mechanism coupled with a system file *(/usr/lib/uucp/L.sys)* and a filesystem permission file *(/usr/lib/uucp/USERFILE)* to control access between systems. The password file entries for *uucp* (usually, *luucp, nuucp, uucp, etc.)* allow only those remote systems that know the passwords for these id's to access the local system. (Great care should be taken in revealing the password for these *uucp* logins since knowing the password allows a system to join the network.) The system file *(/usr/lib/uucp/L.sys)* defines the remote systems that a local host knows about. This file contains all information needed for a local host to contact a remote system (including system name, password, login sequence, etc.) and as such is protected from viewing by ordinary users.

In summary, while the available hardware on a network of systems determines the connectivity of the systems, the combination of *password* file entries and the *uucp* system files determines the extent of the network.

2.3 Forwarding

One of the recent additions to *uucp* (for UNIX System V) is a limited forwarding capability whereby systems that are part of the network can forward files through intermediate nodes. For example, in Figure 1 it is possible to send a file between node *A* and *C* through intermediate node *B*. For security reasons, when forwarding, files may only be transmitted to the *public* area (or fetched from the remote systems *public* area).

2.4 Security

The most critical feature of any network is the security that it provides. Users are familiar with the security that UNIX provides in protecting files from access by other users and in accessing the system via *passwords*. In building a network of processors, the notion of security is widened because access by a wider community of users is granted. Access is granted on a *system* basis (that is, access is granted to *all* users on a remote system). This follows from the fact that the process of sending (receiving) a file to (from) another system is done via daemons that use one special user id(s). This user id(s) is granted (denied) access to the system via the *uucp* system file *(/usr/lib/uucp/L.sys)* and the areas that the system has access to is controlled by another file *(/usr/lib/uucp/USERFILE)*. For example, access can be granted to the entire file system tree or limited to specific areas.

2.5 Software Structure

The *uucp(1)* network is a batch network. That is, when a request is made, it is spooled for later transmission by a daemon. This is important to users because the success or failure of a command will only be known at some later time via *mail* notification. For most transfers, there is little trouble in transmitting files between systems, however, transmissions are occasionally delayed or fail because a remote system can't be reached.

2.6 Rules of the Road

There are several rules by which the network runs. These rules are necessary to provide the smooth flow of data between systems and to prevent duplicate transmissions and lost jobs. The following sections outline what these rules are and what their influence on the network is.

2.6.1 Queuing Jobs submitted to the network are assigned a sequence number for transmission. Jobs are represented by a file (or files) in a common spool directory *(/usr/spool/uucp)*. When a file transfer daemon *(uucico)* is started to transmit a job, it selects a system to contact and then transmits *all* jobs to that system. Before breaking off the conversation, any jobs to be received from that remote system are accepted. The system selected as the one to contact is randomly selected if there is work for more than one system (in releases of *uucp(1)* prior to UNIX System V, the first system appearing in the spool directory was selected so preference was given to the most recently spawned jobs). *Uucp* may be sending to or receiving from many systems simultaneously. The number of incoming requests is only limited by the number of connections on the system and the number of outgoing transfers is limited by the number of ACUs (or direct connections).

2.6.2 Dialing and the DDD Network In order to transfer data between processors that are not directly connected, an auto dialer is used to contact the remote system. There are several factors that can make contacting a remote system difficult.

1. All lines to the remote system may be busy. There is a mechanism within *uucp* that restricts contact with a remote to certain times of the day (week) to minimize this problem.
2. The remote system may be down.
3. There may be difficulty in dialing the number (especially if a large sequence of numbers involving access through PBX's is involved). The dialing algorithm tries dialing a number twice and the algorithm used to dial remote systems is not perfect, particularly when intermediate dial tones are involved.

2.6.3 Scheduling and Polling When a job is submitted to the network an attempt to contact that system is made *immediately*. Only one conversation can exist between the same two systems at a time.

Systems that are *polled* can do nothing to force immediate transmission of data. Jobs will only be transmitted when the system is polled (hourly, daily, etc.) by a remote system.

2.6.4 Retransmissions and Hysterisis The *uucp* network is fairly tenacious in its attempt to contact remote systems to complete a transmission. To prevent *uucp* from continually calling systems that are unavailable, *hysterisis* is built into the algorithm used to contact other systems. This mechanism forces a minimum fixed delay (specifiable on a per system basis) to occur before another transmission can take place to that system.

2.6.5 Purging and Cleanup Users should be aware that transfers that cannot be completed after a defined period of time (72 hours is the value that is set when the system is distributed) are deleted and the user is notified.

2.7 Special Places: The Public Area

In order to allow the transfer of files to a system that a user may not have a login on, the *public* directory (usually kept in */usr/spool/uucp/public)* is available as an area with general access privileges. When receiving files in the *public* area, the user should dispose of them quickly as the administrative portion of *uucp* purges this area on a regular basis.

2.8 Permissions

2.8.1 File Level Protection In transferring files between systems, users should make sure that the destination area is writeable by *uucp*. The *uucp* daemons will preserve execute permission between systems and assign permission 0666 to transferred files.

2.8.2 System Level Protection The system administrator at each site determines the global access permissions for that processor. Thus, access between systems may be confined to only some sections of the filesystem by an administrator.

2.8.3 Forwarding Permissions The forwarding feature is a new addition to the *uucp* package. Users of this feature should understand that,

1. If forwarding is attempted through a node that is running an old version of *uucp(1)*, the transmission will not succeed.
2. Nodes that allow forwarding can restrict the forwarding feature in several ways.
   a. Allow forwarding for only certain users.
   b. Prevent forwarding to certain destination nodes (e.g., nodeblia).
   c. Allow forwarding for selected source nodes.
3. The most important restriction is that forwarding is allowed only for files sent to or fetched from the *public* area.

3. Network Usage

The following sections discuss the user interface to the network and give examples of command usage.

3.1 Name Space

In order to reference files on remote systems, a syntax is necessary to uniquely identify a file. The notation must also have several defaults to allow the reference to be compact. Some restrictions must also be placed on pathnames to prevent security violations. For example, pathnames may not include ".." as a component because it is difficult to determine whether the reference is to a restricted area.

3.1.1 Naming Conventions *Uucp* uses a special syntax to build references to files on remote systems. The basic syntax is,

system-name!path-name

where the *system-name* is a system that *uucp(1)* knows about. The *path-name* part of the name

may contain any of the following,

1. A fully qualified *path-name* such as

   xnode!/usr/you/file

   The *path-name* may also be a directory name as in

   xnode!/usr/you/directory

2. The login directory on a remote may be specified by use of the ~ character. The combination *~user* references the login directory of a user on the remote. For example,

   xnode!~adm/file

   would expand to

   xnode!/usr/sys/adm/file

   if the login directory for user *adm* on the remote system is */usr/sys/adm*.

3. The *public* area is referenced by a similar use of the prefix *~/user* preceding the pathname. For example,

   xnode!~/you/file

   would expand to

   xnode!/usr/spool/uucp/you/file

   if */usr/spool/uucp* is used as the spool directory.

4. Pathnames not using any of the combinations or prefixes discussed above are prefixed with the current directory (or the login directory on the remote). For example,

   xnode!file

   would expand to

   xnode!/usr/you/file

The naming convention can be used in reference to either the *source* or *destination* file names.

3.2 Forwarding Syntax

The newest feature of *uucp* is the ability of *uucp* to allow files to be passed between systems via intermediate nodes. This is done via a variation of the *bang (!)* syntax that describes the path to be taken to reach that file. For example, a user on system *a* wishing to transmit a file to system *e* might specify the transfer as,

uucp file b!c!d!e!~/you/file

if the user desires the request to be sent through b, c and d before reaching e. Note that the pathname is the path that the file would take to reach node *e*. Note also that the destination *must* be specified as the *public* area. Fetching a file from another system via intermediate nodes is done similarly. For example,

uucp b!c!d!e!~/youlfile x

fetches *file* from system *e* and renames it *x* on the local system. The forwarding prefix is the path *from* the local system not the path from the remote to the local system. The forwarding feature may also be used in conjunction with remote execution. For example,

uux xnode!uucp ynode!pnode!/usr/spool/uucppublic/file x

sends a request to *xnode* to execute the *uucp* command to copy a file from *pnode* to *x* on *xnode*.

3.3 Types of Transfers

Uucp(1) has a very flexible command syntax for file transmission. The following sections give examples of different combinations of transfers.

3.3.1 Transmissions of Files to a Remote Any number of files can be transferred to a remote system via *uucp(1)*. The syntax supports the *, ? and *[..] meta* characters. For example,

*uucp *.[ch] xnode!dir*

transfers *all* files whose name ends in *c* or *h* to the directory *dir* in the users login directory on *xnode*.

3.3.2 Fetching Files From a Remote Files can be fetched from a remote system in a similar manner. For example,

uucp xnode!.[ch] dir*

will fetch all files ending in *c* or *h* from the users login directory on *xnode* and place the copies in the subdirectory *dir* on the local system.

3.3.3 Switching Transmission of files can be arranged in such a way that the local system effectively acts as a switch. For example,

uucp ynode!files xnode!filed

will fetch *files* from the users login directory on *ynode*, rename it as *filed* and place it in the login directory on *xnode*.

3.3.4 Broadcasting Broadcast capability (that is, copying a file to many systems) is *not* supported by *uucp*, however, it can be simulated via a shell script as in

```
for i in xnode ynode mhtsd
do
        uucp file $i!broad
done
```

Unfortunately, one *uucp* command is spawned for each transmission so that it is not possible to track the transfer as a single unit.

3.4 Remote Executions

The remote execution facility allows commands to be executed remotely. For example,

uux "!diff xnode!/etc/passwd mhtsd!/etc/passwd > !pass.diff"

will *diff(1)* the password file on *xnode* and *mhtsd* and place the result in *pass.diff*.

3.5 Spooling

If a user wishes to continue modifying a file while a copy is being transmitted across the network, the *-c* option should be used. This forces a *copy* of the file to be queued. The default for *uucp* is not to queue copies of the files since it is wasteful of both cpu time and storage. For example, the following command will force the file *work* to be copied into the spool directory before it is transmitted.

uucp -c work xnode!~/you/work

3.6 Notification

The success or failure of a transmission is reported to users asynchronously via the *mail(1)* command. A new feature of *uucp* is to provide notification to the user in a file (of the users choice). The choices for notification are,

1. Notification returned to the requestors system (via the *-m* option). This is useful when the requesting user is distributing files to other machines and logging onto the remote machine to read mail is inconvenient.

2. A variation of the *-m* option is to force notification in a file (using the -mfile option where file is a file name). For example,

   uucp -mans /etc/passwd ynode!/dev/null

   will send the file */etc/passwd* to system *ynode* and place the file in the bit bucket *(/dev/null)*. The status of the transfer will be reported in the file *ans* as,

   uucp job 0306 (8/20-23:08:09) (0:31:23) /etc/passwd copy succeeded

3. *Uux(1)* always reports the exit status of the remote execution unless notification is suppressed (via the *-n* option). Notification can be sent to a different user on the remote (via the *-nuser* option).

3.7 Tracking and Status

The most pervasive change to the *uucp* package (for UNIX System V) has been revising the internal formatting of jobs so that each invokation of *uucp(1)* or *uux(1)* corresponds to a single job. It is now possible to associate a single job number with each command execution so that the job can be terminated or its status obtained.

3.7.1 The Job ID The default for the *uucp(1)* and *uux(1)* command is *not* to print the job number for each job. This was done for compatibility with previous versions of *uucp(1)* and to prevent the many shell scripts built around *uucp(1)* from printing job numbers. If the following environment variable

JOBNO=ON

is made part of the users environment and exported, *uucp(1)* and *uux(1)* will print the job number. Similarly, if the user wishes to turn job numbers off, the environment variable can be set as follows,

JOBNO=OFF

If the user wishes to force printing of job numbers without using the environment mechanism, the -j option can be used. For example,

```
uucp -j /etc/passwd ynode!/dev/null
uucp job 282
```

forces the job number (282) to be printed. If the -j option is not used, the ids of the jobs belonging to the user can be found by using the *uustat(1c)* command to obtain the job numbers. For example,

```
uustat
0282 tom ynode 08/20-21:47 08/20-21:47 JOB IS QUEUED
0272 tom ynode 08/20-21:46 08/20-21:46 JOB IS QUEUED
```

shows that the user has two jobs (282 and 272) queued.

3.8 Job Status

The *uustat(1)* command allows a user to check on one or all jobs that have been queued. The id printed when a job is queued is used as a key to query status of the particular job. An example of a request for the status of a given job is,

```
uustat -j0711

0711 tom ynode 07/30-02:18 07/30-02:18 JOB IS QUEUED
```

There are several status messages that may be printed for a given job; the most frequent ones are JOB IS QUEUED and JOB COMPLETED that have the obvious meanings. The manual page for *uustat(1)* lists the other status messages.

3.9 Network Status

The status of the last transfer to each system on the network can be found by using the *uustat(1c)* command. For example,

```
uustat -mall
```

will report the status of the last transfer to all of the systems known to the local system.
The output might appear as,

```
znode  08/10-12:35   CONVERSATION SUCCEEDED
mnode  08/20-17:01   CONVERSATION SUCCEEDED
anode  07/22-16:31   DIAL FAILED
nodeb  08/20-18:36   WRONG TIME TO CALL
knode  08/20-20:37   LOGIN FAILED
```

where the status indicates the time and state of the last transfer to each system. When sending files to a system that has not been contacted recently, it is a good idea to use *uustat(1c)* to see *when* the last access occurred as the remote system may be down or out of service.

3.10 Job Control

With the unique job id generated for each *uucp(1)* or *uux(1)* command, it is possible to control jobs in the following ways.

3.10.1 Job Termination A job that may consist of transferring many files from several different systems can be terminated using the *-k* option of *uustat(1)*. If any part of the job has left the system then only the *remaining* parts of the job on the local system will be terminated.

3.10.2 Requeuing a Job The *uucp(1)* package clears out its working area of jobs on a regular basis (usually every 72 hours) to prevent the buildup of jobs that cannot be delivered. To force the date of a job to be changed to the current date, thereby lengthening the time that *uucp* will attempt to transmit the job, the *-r* option can be used. It should be noted that the *-r* option does not impart

immortality to a job. Rather, it only postpones deleting the job during housekeeping functions until the next cleanup.

3.10.3 Network Names Users may find the names of the systems on the network via the *uuname(1)* command. Only the *names* of the systems in the network are printed.

4. Utilities That Use Uucp

There are several utilities that rely on *uucp(1)* or *uux(1)* to transfer files to other systems. The following sections outline the more important of these functions to increase awareness of the extent of the use of the network.

4.1 Mail

The *mail(1)* command uses *uux(1)* to forward mail to other systems. For example, when a user types

mail xnode!tom

the *mail(1)* command invokes *uux(1)* to execute *rmail* on the remote system (*rmail* is a link to the *mail(1)* command). Forwarding mail through several systems (e.g., mail a!b!tom) does not use the *uucp(1)* forwarding feature but is simulated by the *mail(1)* command itself.

4.2 Netnews

The *netnews(1)* command that is locally supported on many systems uses *uux(1)* in much the same way that *mail(1)* does to broadcast *network mail* to systems subscribing to news categories.

4.3 Other Applications

The Office Automation System (OAS) uses *uux(1)* to transmit electronic mail between systems in a manner similar to the standard *mail(1)* command. Some sites have replaced utilities such as *lpr(1), opr(1), etc.* with shell scripts that invoke *uux(1)* or *uucp(1)*. Other sites use the *uucp* network as a backup for higher speed networks (e.g., PCL, NSC HYPERchannel, etc.).

5. Conclusion

This document has outlined the rules that run the network and attempted to illustrate the use and power of *uucp*.

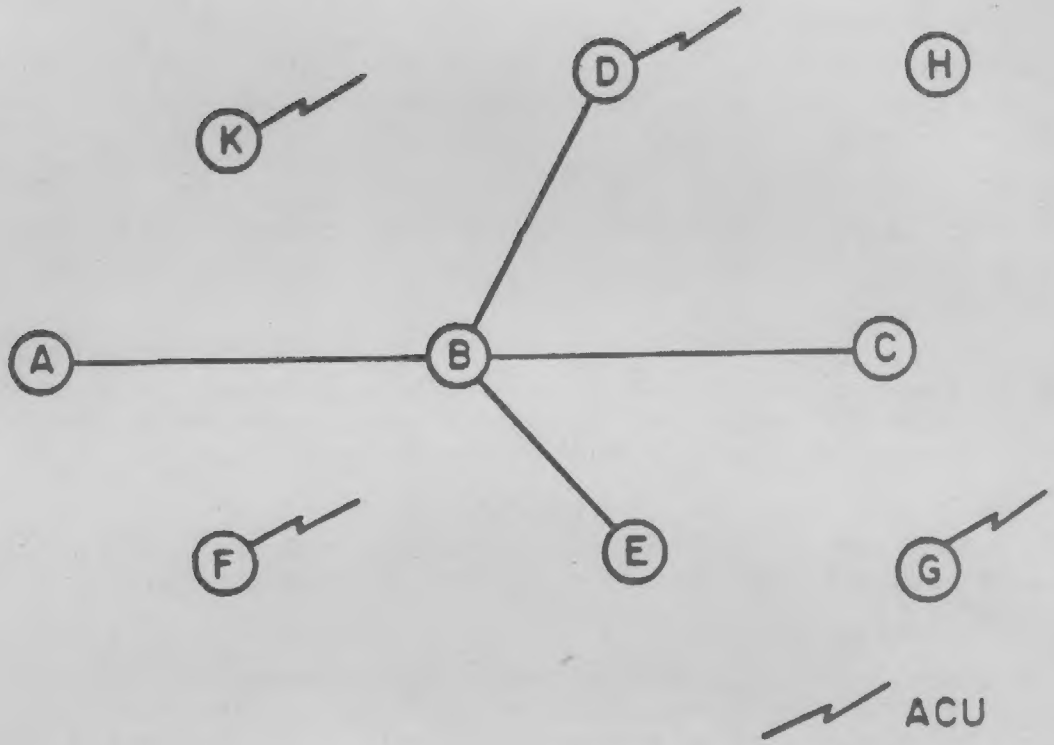


Figure 1 Uucp nodes.

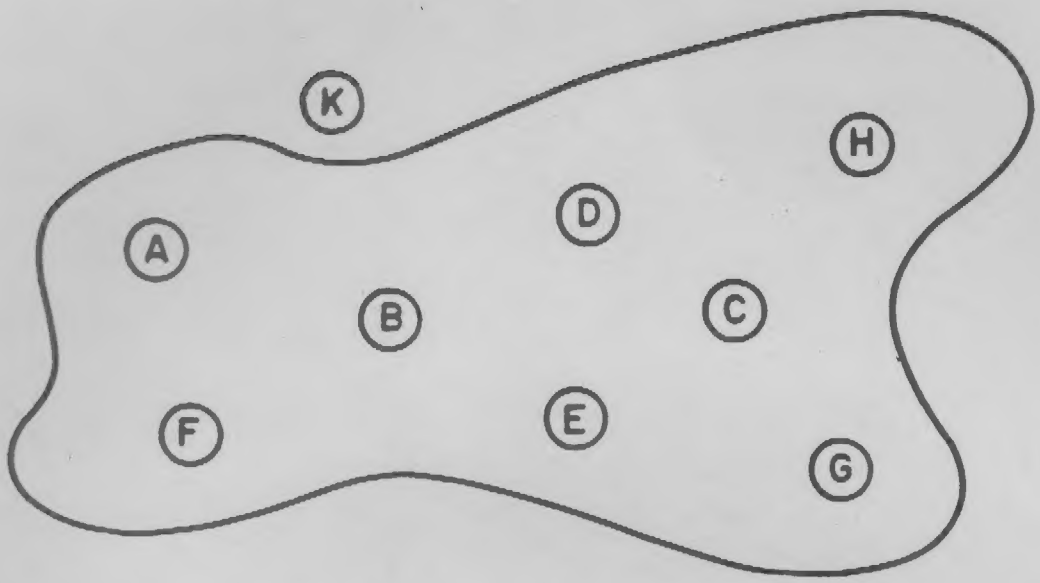


Figure 2a Uucp network excluding one node.

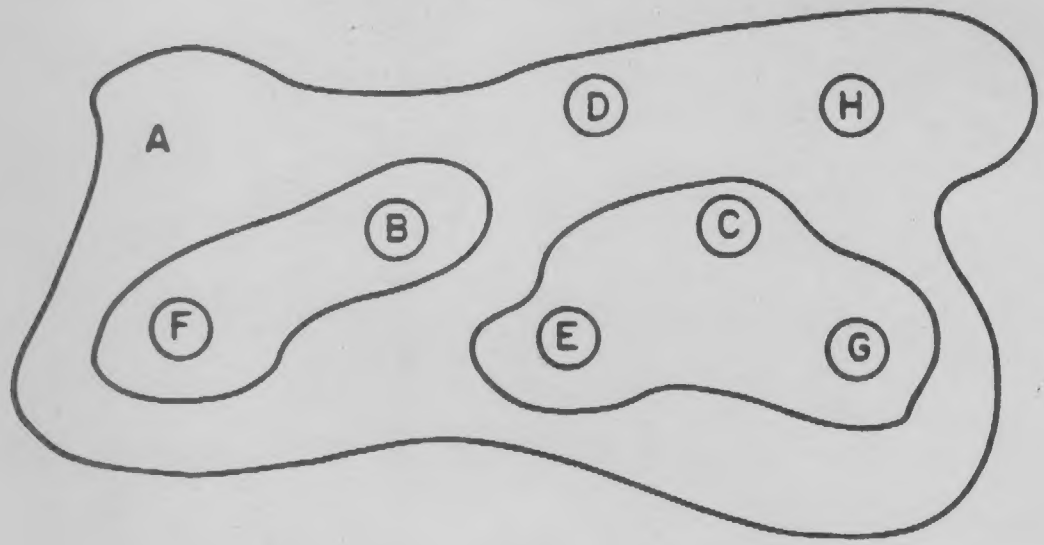


Figure 2b Uucp network with several levels of permissions.

APPENDIX G

Uucp Administrator's Manual

CONTENTS

1. Introduction 1

2. Planning 1
   2.1 Extent of the Network 1
   2.2 Hardware and Line Speeds 1
   2.3 Maintenance and Administration 2

3. A Quick Tour of the Uucp Software 2

4. Installation 2
   4.1 Object Modules 2
   4.2 Password FIle 2
   4.3 Lines File 3
   4.4 System File 3
   4.5 Dialing Prefixes 5
   4.6 USERFILE 5
   4.7 Forwarding File 6

5. Administration 7
   5.1 Cleanup 7
   5.2 Polling Other Systems 7
   5.3 Problems 7

6. Debugging 8

7. Conclusion 8

Uucp Administrator's Manual

1. Introduction

Uucp has been the mainstay of UNIX[1] system to UNIX system communication for several years [1, 2]. This document illustrates how a network is set up, the format of control files and administrative procedures. Administrators should be familiar with the *Uucp Users Tutorial* and the manual pages for each of the *uucp* related commands before reading this document.

2. Planning

In setting up a network of UNIX systems there are several considerations that should be taken into account *before* configuring each system on the network. The following sections attempt to outline the most important considerations.

2.1 Extent of the Network

Some basic decisions about *access* to processors in the network must be made before attempting to set up the configuration files. If an administrator has control over only one processor and an existing network is being joined, then the administrator must decide what level of access should be granted to other systems. The other members of the network must make a similar decision for the new system. The UNIX system *password* mechanism is used to grant access to other systems. The file *(/usr/lib/uucp/USERFILE)* restricts access by other systems to parts of the filesystem tree and the file */usr/lib/uucp/L.sys* on the local processor determines how many other systems on the network can be reached.

When setting up more than one processor is involved, the administrator has control of a larger fraction of the network and can make more decisions about the setup of the network. For example, the network can be set up as a *private* network where only those machines under the direct control of the-administrator can access each other. Granting *no* access to machines outside the network can be done if security is paramount, however, this is usually impractical. Very limited access can be granted to outside machines by each of the systems on the *private* network. Alternatively, access to/from the outside world can be confined to only one processor. This is frequently done to minimize the effort in keeping access information (passwords, phone numbers, login sequences, etc.) updated and to minimize the number of security holes for the private network.

2.2 Hardware and Line Speeds

There are only two supported means of interconnection by *uucp(1)*,

1. Direct connection using a null modem.
2. Connection over the DDD network.

Direct connection over private lines using X.25 is not fully supported in UNIX System V, although with the addition of one program *x25.login* it can be made operational. In choosing hardware, the equipment used by other processors on the network must be considered. For example, if some systems on the network have only 103 type (300 baud) datasets, then communication with them is not possible unless the local system has a 300 baud dataset connected to a calling unit. (Most datasets available on systems are 1200 baud.) If hardwired connections are to be used between systems, then the *distance* between systems must be considered since a null modem cannot be used when the systems are separated by more than several hundred feet (the limit for communication at

---

* UNIX is a Trademark of Bell Telephone Laboratories, Incorporated.

9600 baud is about 800 to 1000 feet although the RS232 specification allows for less than fifty feet). Limited distance modems must be used beyond these distances or if noise on the lines becomes a problem.

2.3 Maintenance and Administration

There is a minimum amount of maintenance that must be provided on each system to keep the access files updated, to insure that the network is running properly and to track down line problems. When more than one system is involved, the job becomes more difficult because there are more files to update and because users are much less patient when failures occur between machines that are under local control.

3. A Quick Tour of the Uucp Software

Figure 1 is an illustration of the daemons used by the *uucp* network to communicate with another system. The *uucp(1)* or *uux(1)* command queues users requests and spawns the *uucico* daemon to call another system. Figure 2 illustrates the structure of *uucico* and the tasks that it performs in communicating with another system. It initiates the call to another system and performs the file transfer. On the receiving side, *uucico* is invoked to receive the transfer. Remote execution jobs are actually done by transferring a command file to the remote system and invoking a daemon *(uuxqt)* to execute that command file and return the results.

4. Installation

The *uucp(1)* package is delivered as part of the standard UNIX system distribution. It resides in its own subdirectory (called *uucp)* in the commands area and has its own make file *(uucp.mk)*. The *uucp* package is installed as part of the normal distribution, however, if it must be reinstalled for any reason, then the sequence

make uucp.mk install

should be executed.

4.1 Object Modules

The following object modules are installed as part of the *uucp* make procedure,

1. *Uucp* - The file transfer command.
2. *Uux* - The remote execution command.
3. *Uucico* - The *uucp* network daemon.
4. *Uustat* - Network status command.
5. *Uuclean* - Cleanup command.
6. *Uusub* - The command for monitoring and creating a subnetwork.
7. *Uuxqt* - The remote execution daemon.
8. *Uudemon.day* - A shell procedure that is invoked each day to maintain the network. Shell scripts for execution each week *(uudemon.wk)* and each hour *(uudemon.hr)* are also distributed.

4.2 Password FIle

To allow remote systems to call the local system, password entries must be made for any *uucp* logins. For example,

nuucp:zaaAA:6:1:45422-UUCP.Admin:/usr/spool/uucppublic:/usr/lib/uucp/uucico

Note that the *uucico* daemon is used for the *shell* and the spool directory is used as the working directory.

4.3 Lines File

The file */usr/lib/uucp/L-devices* contains the list of all lines that are directly connected to other systems or are available for calling other systems. The file contains the attributes of the lines and whether the line is a permanent connection or can call via a dialer. The format of the file is

type line call-device speed protocol

where each field is

type Two keywords are used to describe whether a line is directly connected to another system (DIR) or uses an automatic calling unit (ACU). An X.25 permanent virtual circuit would use the DIR keyword.

line This is the device name for the line (e.g., *ttyab* for a direct line, *cul0* for a line connected to an ACU).

call-device If the ACU keyword is specified, this field contains the device name of the automatic calling unit. Otherwise, the field is ignored, however, a placeholder must be used in this field so that the *protocol* field can be interpreted.

speed The line speed that the connection is to run at. (The speed field is currently ignored if an X.25 link is used.)

protocol This is an optional field that needs only be filled in if the connection is for a protocol other than the default terminal protocol. The X.25 protocol is the only other protocol supported and the single character *x* is used to select this protocol.

The following entries illustrate various types of connections,

```
DIR ttyab 0 9600
ACU cul0 cua0 1200
DIR x25.s0 0 300 x
```

The first entry is for a hardwired line running at 9600 baud between two systems. Note that the *acu-device* field is zero. The second entry is for an line with a 1200 baud automatic calling unit. The last entry is for an X.25 synchronous direct connection between systems. Note that the *protocol* field is filled in and that the *acu-device* and *line speed* fields are meaningless.

4.3.1 Naming Conventions It is often useful when naming lines that are directly connected between systems or which are dedicated to calling other systems to choose a naming scheme that conveys the use of the line. In the earlier examples, the name *ttyab* is used for the line that directly connects two systems named *a* and *b*. Similarly, lines associated with calling units are best given names that relate them to the their calling unit (note the names *cul0* and *cua0* to specify the line and calling unit respectively).

4.4 System File

Each entry in this file represents a system that can be called by the local *uucp* programs. More than one line may be present for a particular system. In this case, the additional lines represent alternative communication paths that will be tried in sequential order. The fields are described below.

system name The name of the remote system.

time This is a string that indicates the days-of-week and times-of-day when the system should be called (e.g., MoTuTh0800−1730).

The day portion may be a list containing some of *Su Mo Tu We Th Fr Sa* or it may be *Wk* for any week-day or *Any* for any day. The time should be a range of times (e.g., 0800−1230). If no time portion is specified, any time of day is assumed to be allowed for the call. Note that a time range that spans 0000 is permitted, for example, 0800-0600 means all times are allowed other than times between 6 and 8 am. An optional subfield is available to specify the minimum time (minutes) before a retry following a failed attempt. The subfield separator is a "," (e.g., Any,9 means call any time but wait at least 9 minutes before retrying the call after a failure has occurred).

device This is either *ACU* or the hard-wired device name to be used for the call. For the hard-wired case, the last part of the special file name is used (e.g., tty0).

class This is usually the line speed for the call (e.g., 300).

phone The phone number is made up of an optional alphabetic abbreviation (dialing prefix) and a numeric part. The abbreviation should be one that appears in the *L-dialcodes* file (e.g., mh5900, boston995−9980). For the hard-wired devices, this field contains the same string as used for the *device* field (e.g., tty0, etc.).

login The login information is given as a series of fields and subfields in the format

[expect send] ...

where *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The expect field may be made up of subfields of the form

expect[−send−expect] ...

where the *send* is sent if the prior *expect* is *not* successfully read and the *expect* following the *send* is the next expected string. (e.g., login--login will expect *login*; if it gets it, the program will go on to the next field; if it does not get *login*, it will send *null* followed by a new line, then expect *login* again.)

There are two special names available to be sent during the login sequence. The string *EOT* will send an EOT character and the string *BREAK* will try to send a *BREAK* character. (The *BREAK* character is simulated using line speed changes and null characters and may not work on all devices and/or systems.) A number from 1 to 9 may follow the *BREAK* for example, *BREAK1*, will will send 1 null character instead of the default of 3. Note that *BREAK1* usually works best for 300/1200 baud lines.

A typical entry in the L.sys file would be

sys Any ACU 300 mh7654 login uucp ssword: word

The expect algorithm matches all or part of the input string as illustrated in the password field above.

4.5 Dialing Prefixes

This file contains the dial-code abbreviations used in the *L.sys* file (e.g., py, mh, boston). The entry format is

abb dial-seq

where

abb is the abbreviation,

dial-seq is the dial sequence to call that location.

The line

py 165–

would be set up so that entry py7777 would send 165–7777 to the dial-unit.

4.6 USERFILE

This file contains user accessibility information. It specifies four types of constraint,

[1] which files can be accessed by a normal user of the local machine,

[2] which files can be accessed from a remote computer,

[3] which login name is used by a particular remote computer,

[4] whether a remote computer should be called back in order to confirm its identity.

Each line in the file has the format

login,sys [c] path-name [path-name] ...

where

login is the login name for a user or the remote computer,

sys is the system name for a remote computer,

c is the optional *call-back required* flag,

path-name is a path-name prefix that is acceptable for *sys*.

The constraints are implemented as follows.

[1] When the program is obeying a command stored on the local machine, the path-names allowed are those given on the first line in the *USERFILE* that has the login name of the user who entered the command. If no such line is found, the first line with a *null* login name is used.

[2] When the program is responding to a command from a remote machine, the path-names allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a *null* system name is used.

[3] When a remote computer logs in, the login name that it uses *must* appear in the *USERFILE*. There may be several lines with the same login name but one of

them must either have the name of the remote system or must contain a *null* system name.

[4] If the line matched in ([3]) contains a "c", the remote machine is called back before any transactions take place.

The line

u,m /usr/xyz

allows machine *m* to login with name *u* and request the transfer of files whose names start with "/usr/xyz". The line

you, /usr/you

allows the ordinary user *you* to issue commands for files whose name starts with "/usr/you". (Note that this type restriction is seldom used.) The lines

u,m /usr/xyz /usr/spool
u, /usr/spool

allows *any* remote machine to login with name *u*. If its system name is not *m*, it can only ask to transfer files whose names start with "/usr/spool". If it is system *m*, it can send files from paths "/usr/xyz" as well as "/usr/spool". The lines

root, /
, /usr

allow any user to transfer files beginning with "/usr" but the user with login *root* can transfer any file. (Note that any file that is to be transferred must be readable by anybody.)

4.7 Forwarding File

There are two files that allow restrictions to be placed on the forwarding mechanism. The format of the entries in each file is the same,

system

or

system!user!user2,...

The file *ORIGFILE (/usr/lib/uucp/ORIGFILE)* restricts the access of systems that are attempting to forward *through the local system*. The file contains the list of systems (and users) for whom the local system is willing to forward. Each entry refers to the system that was the *source* of the original job and not the name of the last system to forward the file. The second file *FWDFILE (/usr/lib/uucp/FWDFILE)* is a list of valid systems that a job can be *forwarded to* (it is not necessarily the name of the destination of a job, but merely the next valid node). This file will be a subset of the L.sys file and can be used to prevent forwarding to systems that are very expensive to reach, but to which access by local users is allowed (for example, links to overseas universities). If neither of these files exist, *uucp* will be perfectly happy to forward for any system. As an example, if the entry for system *australia* were in the ORIGFILE but not in the FWDFILE on system xnode, it would mean that system *australia* would be capable of forwarding jobs into the network via system xnode however, no systems in the network could forward a job to *australia* via system xnode.

5. Administration

The role of the *uucp* administrator depends heavily on the amount of traffic that enters or leaves a system and the quality of the connections that can be made to and from that system. For the average system, only a modest amount of traffic (100-200 files per day) pass through the system and little if any intervention with the *uucp* automatic cleanup functions is necessary. Systems that pass large numbers of files (200-10000) may require more attention when problems occur. The following sections describe the routine administrative tasks that must be performed by the administrator or are automatically performed by the *uucp* package. The section on problems describes what are the most frequent problems and how to effectively deal with them.

5.1 Cleanup

The biggest problem in a dialup network like *uucp* is dealing with the backlog of jobs that cannot be transmitted to other systems. The following cleanup activities should be routinely performed by shell scripts started from *cron(1)*.

5.1.1 Cleanup of Undeliverable Jobs The *uudemon.day* procedure usually contains an invokation of the *uuclean* commany to purge any jobs that are older than some fixed time (usually 72 hours). A similar procedure is usually used to purge any *lock* or *status* files. An example invokation of *uuclean(1m)* to remove both job files and old status files every 48 hours is

/usr/lib/uuclean -pST -pC -n48

5.1.2 Cleanup of the Public Area In order to keep the local filesystem from overflowing when files are sent to the public area, the *uudemon.day* procedure is usually set up with a *find* command to remove any files that are older than seven days. This interval may need to be shortened if there is not sufficient space to devote to the public area.

5.1.3 Compaction of Log Files The files *SYSLOG* and *LOGFILE* that contain logging information are compacted daily (using the *pack* command) and should be kept for one week before being overwritten.

5.2 Polling Other Systems

Systems that are passive members of the network must be polled by other systems in order for their files to be sent. This can be arranged by using the *uusub(1)* command as follows,

uusub -ccnode

which will call *cnode* when it is invoked.

5.3 Problems

The following sections list the most frequent problems that appear on systems that make heavy use of *uucp(1)*.

5.3.1 Out of Space The filesystem used to spool incoming or outgoing jobs can run out of space and prevent jobs from being spawned or much worse received from remote systems. The inability to receive jobs is the worse of the two conditions since when filespace does become available, the system will be *flooded* with the backlog of traffic.

5.3.2 Bad Acu's and Modems The automatic calling units and incoming modems occasionally cause problems that make it difficult to contact other systems or to receive files. These problems are usually readily identifiable since *LOGFILE* entries will usually point to the bad line. If a bad line is suspected, it it useful to use the *cu(1)* command to try calling another system using the suspected line.

5.3.3 Administrative problems Some *uucp* networks have so many members that it is difficult to keep track of changing passwords, changing phone numbers or changing logins on remote systems. This can be a very costly problem since acu's will be tied up calling a system that cannot be reached.

6. Debugging

In order to verify that a system on the network can be contacted, the *uucico* daemon can be invoked from a user's terminal directly. For example, to verify that *cnode* can be contacted, a job would be queued for that system as follows,

uucp -r file cnode!~/tom

The *-r* option forces the job to be queued but does not invoke the daemon to process the job. The *uucico* command can then be invoked directly,

/usr/lib/uucp/uucico -r1 -x4 -scnode

The *-r1* option is necessary to indicate that the daemon is to start up in *master* mode (that is, it is the calling system). The *-x4* specifies the level of debugging that is to be printed. Higher levels of debugging can be printed (greater than 4) but requires familiarity with the internals of *uucico*. If several jobs are queued for the remote system, it is not possible to force *uucico* to send one particular job first. The contents of LOGFILE should also be monitored for any error indications that it posts. Frequently, problems can be isolated by examining the entries in LOGFILE associated with a particular system. The file *ERRLOG* also contains error indications.

7. Conclusion

This manual has emphasized the format of control files and some of the issues in setting up a *uucp* network.

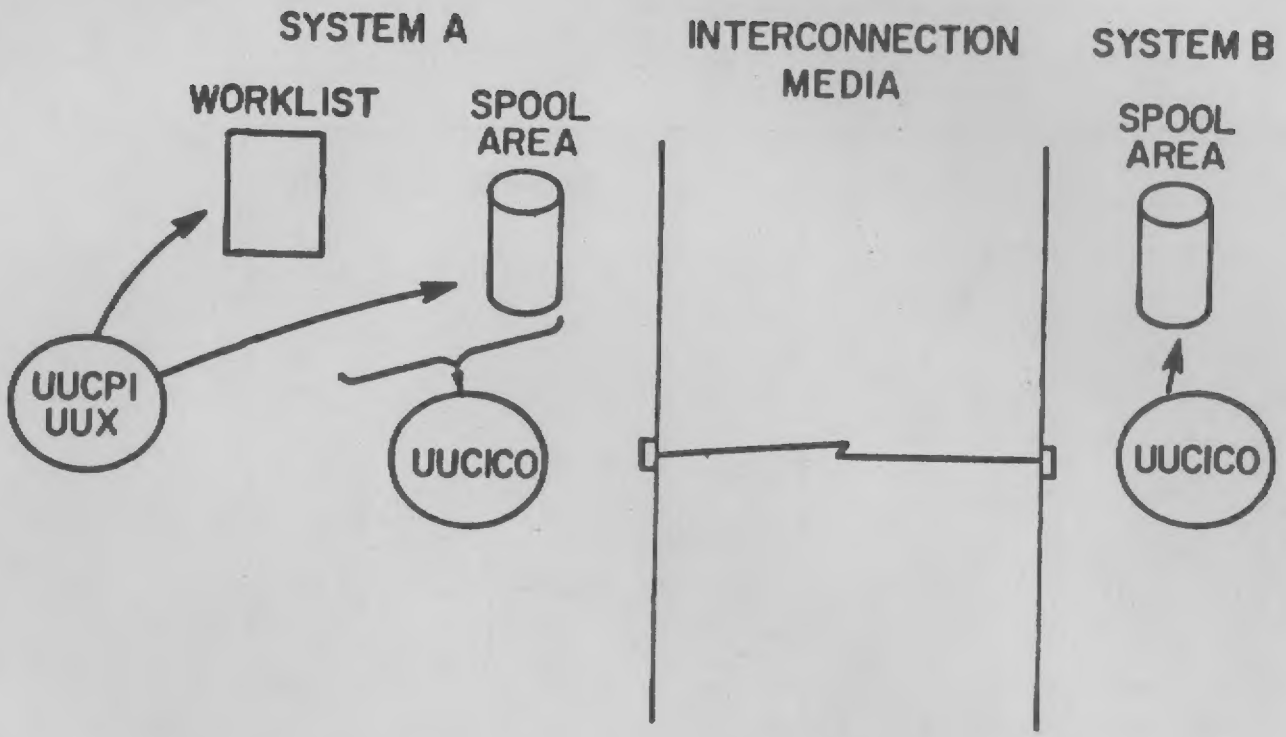


Figure 1 Uucp network daemons.

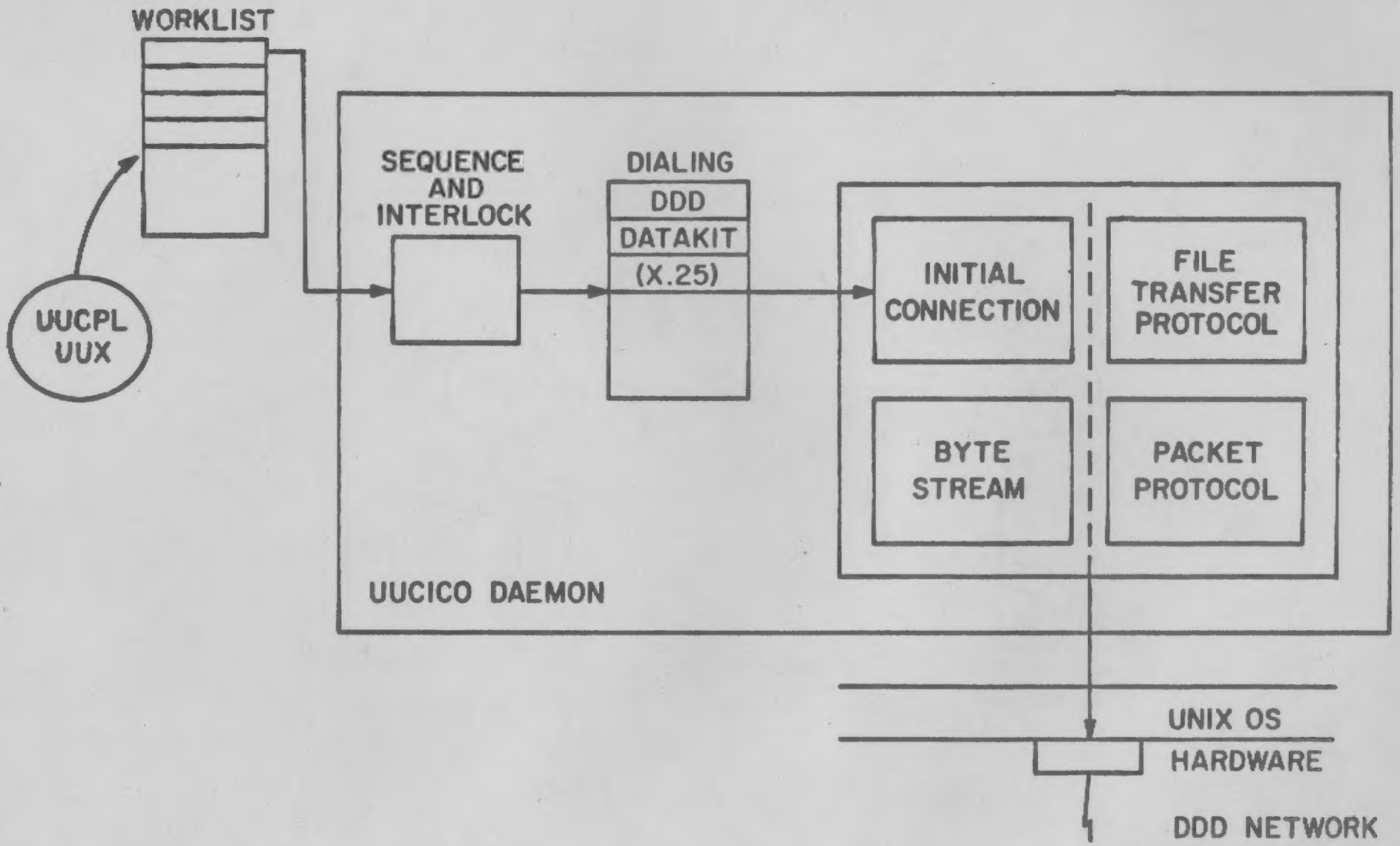


Figure 2 Uucico daemon functional blocks.

APPENDIX H

Installing an Auto Call Facility on a UNIX™ System

Installing an Auto Call Facility on a UNIX™ System

A. INTRODUCTION

You should be thoroughly familiar with the procedures and commands used to install a UNIX[1] system and should have read *Setting up UNIX* before using this memorandum.

The UNIX Time-Sharing System can use a Bell System 801-type Auto Call Unit (ACU) for dialing-up and communicating with other systems. The user-level interface to the ACU is via the *cu* and *uucp* commands. (See *cu*(1C) and *uucp*(1C) in the *UNIX System User's Manual*.) Before ACUs can be used, however, a system administrator must first make certain software adjustments to the system. These adjustments are listed below.

B. PROCEDURES

The following procedures should ensure a properly software-installed automatic call-up facility. If you are installing more than one facility, adjust the instructions accordingly. These instructions apply to DEC UNIX systems and to 3B20 Simplex UNIX systems of Release 5.0 or later. The instructions are divided into a section that applies to both systems, a section on DEC UNIX systems, and a section on 3B20 Simplex UNIX systems.

B.1 Common Procedures

You must have an 801-type ACU and a modem per facility (except as noted below for Bell System 57A1 and 57B1 data units for shared-ACU setups).

1. The ACU must first be optioned correctly, as described below, and associated with a dedicated terminal phone line. For the following examples, **/dev/tty00** will be the dedicated line.

   The 801-type ACU contains the following switches and should be optioned as shown below. A **0**=open and a **1**=closed. (On a closed switch, the side next to the number is down.) Bracketed switches are missing on some models.

   S1 = 1000[1]
   S2 = 0101
   S3 = 11010
   S4 = 11[00]

   If you are using 212-type data sets, make sure they are optioned as follows.

   S1 = [0]001
   S2 = 110001100
   S3 = 11110000
   S5 = 00

   The 57A1 and 57B1 data units are Western Electric devices that allow an ACU to be shared by up to 12 terminal phone lines (data sets). With one 57A1 and one 57B1, a single ACU can dial out on up to six phone lines in one data mounting. With two 57A1s and two 57B1s, a single ACU can dial out on up to twelve phone lines in two data mountings.

---

1. UNIX is a Trademark of Bell Telephone Laboratories, Incorporated.

If you are using the shared-ACU facility, the 57B1 data unit will have to be properly optioned. If the 57B1 data unit is the only sharing circuit in an arrangement, or the first one in an arrangement using two data mountings, all the positions of switches *S9* and *S10* must be open. If the 57B1 is the second sharing circuit in an arrangement using two data mountings, all positions of switches *S9* and *S10* must be closed. If you are using two data mountings, the T and R terminals of TS1 on the two 57A1s must be connected. (See reference 1.) There are no customer-selected options.

2. All make-busy (MB) and service-line (SL) switches on the data mountings should be in the up or off position (i.e., don't busy them out). The make-busy switches on the 57B1s should be in the off position for the slots that have data sets, and in the make-busy position for unequipped or unused slots. Please note the special instructions in *UNIX System Administrator's Guide* when using 103J-type, 113C-type and 113D-type data sets with an 801-type ACU.

   If you have a 212A data set and wish to use it at high speed, the high speed button must be pressed in. This is *not* software switchable. If you choose high speed, any UNIX system you call must have 1200 baud answer capability or you will not be able to connect.

3. Edit **/etc/inittab** and turn the *getty* process *off* for multi-user state (e.g., state 2) by changing the flags field to **off**:

   00:2:off:/etc/getty tty00 1200 #

4. After you complete the steps in one of the following two sections for your hardware, make a new operating system, install it as **/unix** and reboot.

B.2 DEC UNIX Systems

1. You must have a DEC ACU interface, a DN-11 line card. The line card should be optioned for EIA by your DEC Customer Engineer.

2. Include the ACU interface device driver entry in your system description file with the appropriate information for your unit; for example:

   dn11 370 175200 4 1

3. Take note of the major device number generated by **config −t**. It will usually be a **7** on VAX systems and a **4** on PDP-11 UNIX systems. Using the major device number, make the following device entries in the **/dev** directory. The name **dn0** is commonly used with DEC UNIX systems.

   ```
   # mknod /dev/dn0 c 7 0
   # ln /dev/dn0 /dev/cua0
   # ln /dev/tty00 /dev/cul0
   ```

   The linked names of **cul0** and **cua0** are simply aliases of the access and line devices that are known by the *cu* and *uucp* commands.

   In shared-ACU configurations that use two mounting racks, the data sets are numbered 1 through 6 in the first rack and 9 through 14 in the second rack. The data set number should be reflected in the high-order 4 bits of the minor device number of the ACU interface device entries in **/dev**.

   As a comparison, to install four ACU facilities on a UNIX System on a DEC processor would have previously required 4 DN-11 line cards, 4 801-type ACUs, 4 data sets and the following nodes:

Entry	*Minor Device #*
/dev/dn0	0
/dev/dn1	1
/dev/dn2	2
/dev/dn3	3

With the shared-ACU facility, only 1 DN-11 line card and 1 ACU is required. Up to 12 data sets can be serviced and their nodes would be:

Entry	*Minor Device #*	
/dev/dn0	020	(16)
/dev/dn1	040	(32)
/dev/dn2	060	(48)
/dev/dn3	0100	(64)
/dev/dn4	0120	(80)
/dev/dn5	0140	(96)
/dev/dn6	0220	(144)
/dev/dn7	0240	(160)
/dev/dn8	0260	(176)
/dev/dn9	0300	(192)
/dev/dn10	0320	(208)
/dev/dn11	0340	(224)

For more information, see References 1 and 2.

4. Change the modes on the two devices to read/write by all.

   # chmod 666 /dev/cua0 /dev/cul0

   Note that the modes of the aliases of **cul0** and **cua0** are changed automatically because they are links.

5. Ensure that an appropriate entry exists in the file **/usr/lib/uucp/L-devices**.

   ACU cul0 cua0 300

   If you have chosen the high speed switch on the 212A data set, this line should have the speed **1200** instead of **300**.

Note: The above are only examples and should not necessarily be copied directly for your system.

B.3 3B20 Simplex UNIX Systems

1. You must have a 3B20S ACU interface: a TN75B (1-line) or a UN56 (8-line) circuit pack.
2. Specify the interface in the **/etc/system** file. For instance, if you have a UN56 in slot 7 of iop 1, the **/etc/system** file would contain:

```
iop   1      ...
      :
      un56   0   7   0   0   0   0
```

3. Make an entry in the **/dev** directory for each dial-out line. The name **acu** is usually used on 3B20 UNIX systems. The major device numbers are 15 for the TN75B and 25 for the UN56. The following commands make 4 dial-out lines for a UN56.

```
# mknod /dev/acu0 c 25 0
# mknod /dev/acu1 c 25 1
# mknod /dev/acu2 c 25 2
# mknod /dev/acu3 c 25 3
```

4. Change the modes of the device entries to write by all.

```
# chmod 222 /dev/acu[0-3]
```

5. Make sure that the appropriate entries exist in the file **/usr/lib/uucp/L-devices**. For example, if the 4 dial-out lines are associated with terminal devices 50 through 53, the entries would be:

```
ACU tty50 acu0 1200
ACU tty51 acu1 1200
ACU tty52 acu2 1200
ACU tty53 acu3 300
```

The last column specifies baud rates. These should be 1200 if the high-speed switch on the data set is depressed, 300 if not.

6. Make sure that the modes of the terminal lines are read/write by all:

```
# chmod 666 /dev/tty5[0-3]
```

7. If you are using a UN56, create a file that specifies the ACU configuration to the UN56. This file is used by the *acuset*(1M) command to initialize the UN56. The file is made up of lines of the form:

```
/dev/acu? unit port [slot]
```

where */dev/acu?* is a device node as created above, *unit* is the UN56 unit number, *port* is the port on that unit (marked on the distribution panel), and *slot* is the slot number in a sharing arrangement (1-6 for the first data mounting, 7-12 for the second). A non-shared ACU does not have a slot number. For example, the following file specifies that the four devices created above are attached to UN56 unit 0. The first two are in slots 1 and 2 in a sharing arrangement on port 0; the third is in a non-shared arrangement on port 1, and the fourth is non-shared on port 2.

```
/dev/acu0 0 0 1
/dev/acu1 0 0 2
/dev/acu2 0 1
/dev/acu3 0 2
```

See the *acuset*(1M) manual entry for more information.

8. The UN56 will not work unless it is initialized. Change the **/etc/rc** file to call the **acuset** command with the configuration file created above to initialize the UN56. For example, if the file is named **acus**, add the line

```
/etc/acuset acus
```

to the **/etc/rc** file.

C. DIAGNOSING PROBLEMS

If the above steps are followed precisely and the unit still does not work, the hardware should be checked out. Problems should be diagnosed in the following order:

1. ensure that the lock file (**/usr/spool/uucp/LCK...**) is not present from earlier failed attempts with *cu* or *uucp*,

2. perform the self-tests on the data sets, the ACUs, and the sharing hardware (these tests are described in literature that comes with the devices),

3. verify that both the ACU and the data set are correctly optioned as described above,
4. check that the ACU interface device (UN56, TN75B, or DN-11) is pulsing digits to the ACU,
5. determine that the ACU is dialing the *correct* digits,
6. ensure the integrity of the data set by using it as a dial-in port, and
7. determine that the cable leads are not defective.

D. REFERENCES

[1] *UNIX™ System User's Manual - System V.*

[2] *UNIX™ System Administrator's Manual - System V.*

APPENDIX I

UNIX™ System Modification Requests
Resolved in UNIX™ System V

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
1	/etc/crontab	disk accounting misses protected files	mb81-21109
2	/etc/group	/etc/group must take out blank lines	bl82-17502
3	/etc/master	allow prf_cnt	bl81-04801
4	/etc/master	change default disk	bl81-03902
5	/etc/profile	logname variable checked for mail	bl81-22217
6	/etc/profile	should get rid of readonly attribute of logname.	bl81-22216
7	/etc/update	the /etc/update command is undocumented.	bl81-29001
8	/usr/include	different assignment of error numbers in errno.h and sys/user.h	bl80-19012
9	/usr/include/a.out.h	inappropriate field type in r_info struct	bl80-35209
10	/usr/include/sys/tty.h	incompatible flags	bl81-01617
11	/usr/lib/acct/ckpacct	options to date command misunderstood by sccs	bl82-07302
12	/usr/lib/acct/ckpacct	update ckpacct for new block size	bl82-07301
13	/usr/lib/crontab and /etc/rc	entries to invoke sa1,sa2 and sadc in crontab, rc files are missing.	bl81-31013
14	/usr/lib/uucp/l.sys	blank line in l.sys file causes uucp to blow up	bl81-04401
15	/usr/lib/uucp/l.sys	new uucp names	bl81-16809
16	/usr/src/cmd/bc/bc.c	compiler produced warning messages.	bl80-34403
17	/usr/src/cmd/date.c	change to conform to new time.h and ctime.c	bl80-21011
18	/usr/src/cmd/df.c	df dumps core if no args are given	bl81-34306
19	/usr/src/cmd/errpt.c	incompatibility with the c-library.	bl80-21015
20	/usr/src/cmd/text/macros.d/ntar.c	compiler generated warning.	bl80-35001
21	/usr/src/cmd/text/neqn.d/io.c	compiler produced warning.	bl80-35004
22	/usr/src/cmd/touch.c	incompatibility with c-library.	bl80-21014
23	/usr/src/uts/pdp11/os/sys1.c	the maxproc limit is not enforced on set uid processes.	wc80-27507
24	02/15/81	arcv has lint errors.	bl81-02802
25	11/70	etp 2.0 does not work.	up82-10403
26	3.0 manual	return value for abort (3c) not documented.	bl81-10312
27	:mk	include :mksyshead	bl80-33610

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
28	:mk*head	modes for directories should be 755	bl82-17304
29	:mkcmd	need fix for redundancy	bl81-32714
30	:mkhead	:mkhead has a syntax error	bl82-12801
31	:mkhead	should make owner bin on all header files.	bl81-30106
32	:mksyshead	the :mkhead and :mksyshead makefiles should not include 370	bl82-15801
33	:mkuts	:mkuts should be made portable	bl82-06807
34	:mkuts	see below.	bl81-07309
35	<std10.h	4.0	bl80-25606
36	a.out.h	"value" field in symbol-table entry should be unsigned.	ua82-06901
37	a.out.h	don't change a.out.h header file for release 5.0.	bl82-04002
38	a.out.h	fix "octal" 8's	bl81-22606
39	a.out.h header file	causes incorrect error messages from lint.	bl81-31006
40	a.out.h,nm	n_type char -	bl81-00914
41	accounting	need source independent holiday array for accounting	bl82-15916
42	accounting system	word boundry alignment problems with scanf in accounting programs	bl81-21211
43	acct	provide additional accounting data.	at80-32419
44	acct	turnacct switch eats up /usr	bl80-26004
45	acct.mk	acct.mk doesn't install holidays table file	bl82-18010
46	acct.mk	support for no floating point.	bl80-32058
47	acctcom(1)	acctcom skips records when reading from a pipe.	bl82-07510
48	acctcom(1)	exit status gets sign extended	bl81-02908
49	acctcon1	acctcon1 doesn't print connect accounting usage	bl82-16903
50	acctcon1(1m)	acctcon1 produces bad data	bl82-22312
51	accting	update accounting for init.	bl82-09005
52	acu	can not diconnect from cu	cc82-07003
53	acu access	general dialup routine.	bl80-34703
54	acuset	acustat, acusdev, acustart undefined in acuset.c.	bl82-22111
55	adb	5.0 adb will not build on the vax	bl82-10303
56	adb	adb breakpoints truncate addresses to 16-bits (insufficient on vax)	bl81-11806
57	adb	adb interprets signals wrong	bl80-32219
58	adb	adb is obsolescent on the vax, the man page should say this.	bl82-03202
59	adb	adb print in decimal.	bl80-21207
60	adb	d format no longer works.	bl80-21306
61	adb	fails with pre-release with large kernel text.	bl80-23905

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
62	adb	print in unsigned decimal.	bl80-22001
63	adm files	uucp administartive files must be cleansed.	bl82-33501
64	adm. manual	introduction cross reference.	bl81-20908
65	admin	admin declares all files that $ in their name to be corrupted.	bl80-04417
66	admin	problems with admin	bl80-35304
67	admin -z	the admin -z command can randomly delete the s.files	bl81-01315
68	admin advice document	dn11 acu options should be mentioned	mb81-27002
69	admin command	admin should not accept file names longer than 14 characters	bl80-00312
70	admin. advice document	212ar-l1 modem options wrong	mb81-21903
71	admin. manual	pnch file information in rje (8) belongs in user's manual.	bl81-22623
72	administrative advice	dzb: 77 17 177777 177777 not documented	bl81-32921
73	administrative advice	include steps to install acu	bl79-12003
74	administrative advice for unix/ts	212ar data set options prevent catching hangup signal.	bl80-32504
75	all	name and initials in source.	bl82-28109
76	all	update global makefile	bl81-24001
77	all over	clean-up sccs source	bl80-14108
78	all over	decrease use of the pw library.	bl80-15609
79	aouthdr.h	add vstamp to aouthdr.h	bl82-06408
80	ar	ar does not pad to a word boundary on the vax.	bl80-17803
81	ar	archieve format truncates user id and group id.	bl80-21004
82	ar	the 4.0 ar does not coincide with manual page.	bl81-02803
83	ar (5) manual page.	the ar(5) manual page is wrong for the vax-1170	bl81-21609
84	ar(1)	fix compiler warning	bl81-32005
85	ar.h	the ar header has the wrong type for ar_date.	bl81-01304
86	arcv	arcv builds max million plus file	bl80-20714
87	as	"common" as software not truly common.	cc82-02502
88	as	.comm pseudo op added for f77	uz82-05401
89	as	.long bug was found in the assembler.	bl82-12309
90	as	add fp move round and change move trunc. instrs.	bl81-25317
91	as	assembler handles address calculations improperly.	ua82-05401

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
92	as	assembler needs flag to make data read-only.	bl81-11104
93	as	change relocation pointer for user-defined functions.	bl81-31710
94	as	common as is too slow.	cc82-01504
95	as	correct the float.c file in the asm.	bl81-23013
96	as	implementation documentation was wrong for movcce instruction	bl82-06405
97	as	increase ninstrs	bl81-31607
98	as	new opcodes for dmert were added	bl81-29660
99	as	oneproc.mk does not do make install properly	uz82-05313
100	as	relocation pointers -	bl81-35110
101	as	several instructions specified by dec aren't available from the assembler.	bl80-25611
102	as	the assembler was setting up bit fields wrong.	bl82-11807
103	asa/fsplit	install new fortran utility programs asa and fsplit	bl80-32053
104	asndev	asndev command is documented as asgndev.	bl80-31245
105	assert (3)	improvements to assert.h header file.	bl81-10310
106	asych. comm. microcode	asserts dtr falsely	bl82-11804
107	atb	need to use all atbs	bl81-23609
108	atof	atof returns undefined value if only new line is presented.	bl80-26605
109	atof(3c)	ascii convert functions should report failure	bl80-24014
110	atof(3c)	provide atox function	bl80-24013
111	awk	awk can't add.	bl82-08505
112	awk	awk has local pathnames built in	bl82-06914
113	awk	awk makefile does not work	bl82-18805
114	awk	awk program works on rel 2.0 and core dumps on 3.0 vax.	bl80-27513
115	awk	compiler produced warning messages.	bl80-34401
116	awk	connect awk with the shell environment for use in passing parameters	bl80-14412
117	awk	core dump when setting $1 = to a long (60) string of 9's	bl81-34309
118	awk	length() returns floating point value.	bl81-31529
119	awk	makefile for awk (awk.mk) not correct.	bl80-22007
120	awk	printf "%%" works incorrectly in	bl81-03321

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
		3.0	
121	banner.c	incorrect argument supplied to function banfil()	bl81-21201
122	base regs	extra base register enhancement on 11/70	bl80-25411
123	basename	basename (1) is inconsistant.	bl81-22634
124	basename	basename and dirname - three bugs	bl81-25202
125	basename	on a no match with a suffix, basename returns a blank.	bl81-33611
126	basic system files	see below.	bl81-07304
127	bc	faulty exponentiation (bc)	bl81-04407
128	bc(1)	18%5 gave 0 when executed under "bc -l" command.	bl81-34106
129	bc.mk	discard bc.x	bl81-08105
130	bfs	bfs core dumps on "e file"	bl82-17310
131	bfs	incorrect handling of shell escapes	bl80-21001
132	bfs	prompt option has gone bad	bl82-17311
133	boot	cannot boot a pdp11/70 with rm05 disk drives.	up82-12001
134	boot	it doesn't work.	bl81-08216
135	boot loader	distributed unix boot loader is too small for systems with many devices.	bl80-32074
136	boot programs	dump rk05 as a boot device.	bl80-27604
137	boot programs	remove rf11boot	bl81-07308
138	boot/iltd	provide generalized disk boot	bl81-04005
139	booterr.8	booterr.8 man page is incorrect	bl81-30912
140	bootload	disk boots do not work with all dec console floppys.	bl81-08407
141	bprintf, bwritef	_doprnt needs to be replaced with sprintf	bl82-22202
142	brk(2) and sbrk(2).	brk() and sbrk() gave memory fault. og=st;	bl82-01002
143	bs	arithmetic error	bl81-08515
144	bs	extend dump command in bs (bs.c, bs.1)	bl80-32041
145	bs(1)	2	bl81-35004
146	bs(1)	bs(1) core dump.	bl81-22624
147	bs(1)	bs(1) loops and ignores signals.	bl81-22628
148	bs(1)	bs(1) treats exponentiation illogically.	bl81-23009
149	bs(1), sno(1)	bs(1) and sno(1) manual pages are missing.	bl81-22616
150	bsh	set -e applies to tests in if statements.	bl78-18121
151	buffer limit	extend the buffer limit to 2048	bl81-30136
152	builds	incomplete makefiles	bl80-35305
153	builds	plug up security holes	bl81-07901

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
154	c	defective syntax equation for statement	bl79-19102
155	c	wrong answer to the trig functions.	bl81-27511
156	c compiler	(char) is treated as an (int)	bl80-22408
157	c compiler	+= in argument	bl81-25706
158	c compiler	add a c compiler which accepts multiple definitions of external symbols.	bl82-23016
159	c compiler	bad code with register variables	bl80-20601
160	c compiler	bug - "correct" program gives "incorrect" results	bl81-30908
161	c compiler	c compiler apparently fails to cast to (int).	bl80-11405
162	c compiler	compilation error have to register char variables in function argument list.	bl81-02217
163	c compiler	conversion of long to short does not mask out high-order bits.	mh82-20406
164	c compiler	does not generate cc -g symbol table information for bit field	bl80-32060
165	c compiler	error in indirect access to low-order half of a "double".	ua82-05301
166	c compiler	field assign float	bl81-25703
167	c compiler	incorrect compilation of boolean expressions	bl81-19010
168	c compiler	initialization of pointer to void function generates error message.	bl81-01909
169	c compiler	multiple double precision assignments	bl81-25715
170	c compiler	pseudo-arguments accepted in external function definition.	bl80-25505
171	c compiler	sizeof operator doesn't work	bl81-24601
172	c compiler	update 'c' compiler and fix bugs.	bl80-32036
173	c compiler optimizer	adopt new version	bl80-32035
174	c compiler optimizer	c2 removes live scratch register	bl82-11101
175	c complier		bl81-17610
176	c lib	the format %ld handles negative numbers incorrectly.	bl82-08106
177	c lib/usr/include	add new rutine to c-lib (and new .hfile/usr/include)	bl81-02807
178	c library	3 problems with the c-library routines that make temp-files	bl81-03708
179	c library	add strtol.c routine to the c-lib.	bl81-02806
180	c library	buffering scheme in c library stdio needs to be restructured for increased speed and relia	ua82-02501
181	c library	cc -p should link to a library	bl82-02913

NR	AREA	SUMMARY	MR
		which gives call-counts for function calls.	
182	c library	doprnt.s must be run through cpp during make. this should be changed.	bl82-07109
183	c library	format " %d" behaves incorrectly when eof encountered on terminal.	bl82-05402
184	c library	functions are needed to manipulate memory areas regardless of content.	bl81-26511
185	c library	functions that loop on getc or putc are grossly inefficient.	bl81-26512
186	c library	libc/port/stdio/doscan.c is too slow.	bl82-07111
187	c library	nlist.c needs to be changed to work with the 5.0 common object file	bl82-04809
188	c library	pdp11 c library assembly language functions do not support profiling	bl82-04714
189	c library	stdio performance can be improved.	bl82-07406
190	c library	the code for handling dummy file numbers for sscanf is dangerous.	bl82-07802
191	c library	the first time mcount is called it dumps core.	bl82-07114
192	c library	the port directory makefile cannot run to completion on pdp11/70's.	bl82-10408
193	c library	the profiled libraries are not being optimized.	bl82-07110
194	c library	various fakcu.s routines should be replaced by 1 c function fakcu.c	ua81-32001
195	c library - tsearch	several bug fixes for release 4.0	bl80-36402
196	c library and math library	mrs for library changes were entered as enhancements; should have been fixes.	bl82-03402
197	c library drand48	enhanced random-number generator family drand48.	ua82-02601
198	c preprocessor	comments can confuse it when lint invokes it	bl81-25902
199	c programs that use pwb pre-processor symbol	pwb symbol is obsolete.	bl80-20713
200	c start up	c start up incorrect for 5.0	mh82-11602
201	c-compiler	dimension table overflow	bl81-04313
202	c-compiler	upgrade compiler for 4.2 release.	bl81-23001
203	c-compiler?	minus infinity incorrectly returned by log function.	bl81-27508
204	c-lib	fopen + fseek + fgets work improperly.	bl82-02915
205	c2	new optimizer is required due to	bl81-22651

NR	AREA	SUMMARY	MR
		changes in compiler output.	
206	c2	optimizer can get confused here functions begin	bl82-12501
207	calendar	7/24 is no longer picked up on july 24	bl80-22513
208	calendar	calendar does not accept month with leading zero absent. have solution.	bl80-18209
209	calendar	calendar invokes mail even if there are no calendar entries.	bl81-01902
210	calendar	calendar invokes mail even if there is no calendar entries	bl81-01603
211	calendar	calendar no longer recognizes dates in thr form of 01/17	bl80-22402
212	calendar	egrep complains if no $home/calendar	bl81-32704
213	calendar	extend format of dates recognized.	bl80-22009
214	calendar	numeric months that don't begin with 0	bl80-32008
215	cassi	mesaid is updated in msp field wich is useless.	bl81-21207
216	cat	'cat' command does not handle error correctly.	bl81-31314
217	cat	cat doesn't flush buffers before giving an error message.	bl80-26604
218	cat	the form cat a b	bl80-18213
219	cb	cb returns arbitrary exit code	bl80-32043
220	init	can't go multi-	mh82-12102
221	init	init gets locked up	mh82-12101
222	init	system trys to go multi-user on initial boot.	mh82-20401
223	init	init does not work on pdp11 or vax750	bl82-14002
224	init	wall runs totally asynchronously...	bl82-15201
225	init	entries for brc, bcheckrc, and gettydefs are wrong	bl82-13911
226	init	gettydefs should allow comments	bl82-16108
227	init	gettydefs should have 1200 baud as default	bl82-17623
228	init	init hangs if you go the improper multi-user state	bl82-14608
229	init	init.mk needs to link /etc/init to /bin/telinit	bl82-15202
230	init	initdefault state does not work as advertised	bl82-14607
231	init	inittab, rc, shutdown, brc..... need to be fixed to be compatible	bl82-15204
232	init	powerfail messages should go to /dev/console	bl82-16106

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
233	init	shutdown and cbinit incompatible	bl82-15203
234	init	the .adm makefile needs to be updated for cb init features	bl82-13708
235	init	who and whodo must be made more compatible with former releases	bl82-11703
236	init/getty	initial login message sends junk parity bits.	ua82-10202
237	init/getty	getty changes erase delete and eol characters on login	bl82-15903
238	init/getty	printing node name info causes terminal to hangup	bl82-15901
239	cc	a form feed beginning a line claims the first character position	bl79-04301
240	cc	a while loop controlled by a register unsigned short var does not stop on vax	bl80-34713
241	cc	abstract=compiles floating-point constants incorrectly.	bl82-09904
242	cc	c compiler symbol table too small	mh82-14801
243	cc	cc does not take combined options	bl81-22706
244	cc	cc dumps core using certain options.	bl82-10204
245	cc	cc fails - "too many files"	bl81-28601
246	cc	cc merged for all machines.	bl82-03206
247	cc	cc rejects register char declaration in a function.	bl82-04203
248	cc	conflict in use of -d option	bl80-25610
249	cc	glitch in handling *void* non—type.	bl81-14501
250	cc	improper implementation of precision rules.	bl81-20317
251	cc	improve handling of floating-point expressions.	ua82-10203
252	cc	incorrect code generated for "register char" variable.	bl80-06002
253	cc	integer right shift	bl81-11401
254	cc	more than 250 tokens blow the c compiler	ms81-24319
255	cc	optimizer screws up.	bl80-03704
256	cc	over ambitious optimization	mh82-21501
257	cc	silly conversion code generated should be optimized out	bl82-09709
258	cc	the -b option of the cc command should not have to end with "/".	mh82-06002
259	cc	vax c compiler bug.	bl81-32907
260	cc (1) ld (1)	cant send -t option to ld (1).	bl81-20407
261	cc -o	vax optimizer (c2) breaks code around "extzv" instruction	ua82-24601
262	cc cmd	limit of .c files	bl81-25313

NR	AREA	SUMMARY	MR
263	cc cms	add -ln flag for as	bl81-24417
264	cc code generator	mucks up register variables as function arguments.	bl81-30901
265	cc command	-i option to cc command creates havoc with following option	uz82-07202
266	cc command	-lc last on ld command line	bl82-06808
267	cc command	-m flag	uz82-07401
268	cc command	-n loader flag	uz82-08101
269	cc command	assembler escapes caused errors	cc82-08802
270	cc command	cc command must recognize both -i and -ild	bl82-06112
271	cc command	combination of -c and -p options causes cpp mixup	bl82-19615
272	cc command	forgot to declare functions	uz82-05304
273	cc command	have cc command access profiled libs, include debug flag	bl82-05703
274	cc command	new cc command mishandles -l loader flags	uz82-05310
275	cc command	new flag: -m	uz82-08001
276	cc command	reorder loader line	uz82-08401
277	cc command	the cc command -v (version) option was expecting an	uz82-07801
278	cc man page	the -u description misleading	bl80-29409
279	cc(1)	empty mon.out file	bl81-24603
280	cc, pcc	pointers to functions returning void not handled correctly	bl82-09703
281	cc/lint	duplicated typedefs blow-up lint/pcc parser.	bl80-35802
282	cc/pcc	the cc and pcc compilers differ.	bl80-18413
283	cd	the cd(1) man page does not document $cdpath	bl81-21802
284	cf.mk	cf.mk needs to work with new vpm	bl82-11306
285	cf.mk	leaves a.out around	bl81-06706
286	cf.mk	make it release 4.0	bl80-29701
287	cf.mk	security holes	bl81-08103
288	cf.mk	security nit.	bl81-03101
289	cf.mk	sysfix is too verbose	bl81-06705
290	cfimk	shouldn't remove a.out if sysfix error.	bl80-26912
291	cflow	add the 'cflow' program to unix release 4.0.	bl80-32050
292	cflow	doesn't handle .s or .o files	bl81-17614
293	cflow	file names in the flow graph are restricted to	bl81-16402
294	cflow	misplaced auxiliary files.	bl80-34605
295	cflow(1)	options are undocumented	bl81-08002
296	change management	change management for 5.0 has not been audited	bl82-06317
297	checkall	adopt new command checkall to speed up file system checking	bl81-01514

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
298	checkall command	checkall looping	bl81-03516
299	chmod and chown system calls	chmod and chown system calls work incorrectly with open files on a read-only file system	bl80-27704
300	ckfile	ckfile should recognize vax magic numbers	bl82-23611
301	clib documentation	official documents should be updated.	bl81-19902
302	clock.c	add new library function clock(3c) for simple interface to time(2).	bl82-02918
303	clock.c : timeout();	nondetectable callout table overflow.	bl81-15309
304	clri	clri will not compile on the pdp 11/70	bl82-16112
305	cmd	change kasb to agree with new sgs	bl82-10301
306	cmd	sys/signal.h -	bl81-02910
307	cmds	some commands switch a long instead of an int.	bl82-12009
308	cmp(1)	cmp return code misleading	bl80-29533
309	cmts data base	discrepency of initials for names in the data base.	bl82-03407
310	comb	comb -s assumes incorrectly.	bl80-13609
311	comb.	change to delta compress function.	bl80-03503
312	commans	modified commands need to be integrated into standard ones	bl81-21704
313	comment scrubbing	remove btl (not bsd) references from comments	bl82-26008
314	comp	profiled compiler core dumps with illegal instruction	bl82-06709
315	comp	remove block internal declaration bug fix	bl81-26202
316	comp	sid in improper format	bl81-24031
317	compiler	add floating point	bl81-23207
318	compiler	compiler generating bad code.	bl82-10203
319	compiler	compiler is not converting short types correctly.	bl82-14507
320	compiler	depndncy missing in makefile	bl81-24418
321	compiler	floating point	bl81-23208
322	compiler	init for static floats	bl82-12604
323	compiler	missing movtwb instruction	bl81-29638
324	compiler	new float instructions	bl81-25318
325	compiler	pcc r.1.1	bl81-23824
326	compiler	unsigned short	bl81-32012
327	compiler dimension table	compiler dimension table too small	bl80-21408
328	compiler and optimizer	correct floating point instructions in compiler and optimizer	uz82-06401
329	compiler and optimizer	tmpfile redefined	bl82-06703

NR	AREA	SUMMARY	MR
330	conf.c	can't exclude sema or messages	bl81-19103
331	config	bad default for an option.	bl81-02817
332	config	bad default option for config.pdp.c	bl81-03709
333	console terminal	provide cntl/o type facility for vax	bl80-33201
334	convert	there are problems with the convert command.	bl82-18201
335	cp	cp does not check argument count.	bl80-27707
336	cp	does not work as 3.0 cp worked	bl81-07521
337	cp	the 'cp' command does not detect all instances of bad arguments.	bl81-03734
338	cpio	'b' option being used when 'passing' files	bl81-12409
339	cpio	allow cpio to do file read/write's in biger byte blocks.	bl80-30805
340	cpio	change of manual page for cpio.	bl80-30801
341	cpio	copy of a file that already exists.	bl81-05106
342	cpio	cpio -p option doesn't work with special files	bl80-22813
343	cpio	cpio does not make directories.	bl81-16904
344	cpio	cpio does not reset modes on already existing files	ua82-00702
345	cpio	cpio enters an infinite loop	bl80-31602
346	cpio	cpio exits if it detects a directory and -d option was not specified.	bl81-05017
347	cpio	cpio fails to link full pathnames	bl80-32079
348	cpio	cpio fails to remake already existing nodes even with -u option	ua82-07002
349	cpio	cpio has problems with piped input	bl81-34320
350	cpio	cpio loops if it can't create a file	up82-11902
351	cpio	cpio moves files when it needn't	bl80-31807
352	cpio	cpio prints number of blocks copied in 512-byte blocks	bl80-31238
353	cpio	cpio should notify if 'd' option is forgotten.	bl80-32056
354	cpio	cpio shouldn't copy if mod. times are the same.	bl80-32055
355	cpio	empty directories not created	bl81-35226
356	cpio	enhancement of swap option in cpio.	bl80-30803
357	cpio	missing newline	bl80-23821
358	cpio command	cpio -pd hangs if no write permission on copied directory	bl81-33407
359	cpio error messages	cpio uses filenames in diagnostics when it should use pathname	bl80-24717
360	cpio(1)	bad -i explanation of cpio(1)	bl82-02211

NR	AREA	SUMMARY	MR
		command.	
361	cpio(1)	cpio still loops endlessly on mkdir trouble	bl81-32017
362	cpio(1) and find(1)	cpio (1) -b option does not work for tapes.	bl81-22626
363	cpio(5)	errors/omissions in description of cpio file format	bl81-13903
364	cpp	c preprocessor #if can't handle comments.	bl81-18109
365	cpp	cpp bug when passing long, ifdef'd out comments	bl81-18002
366	cpp	create manpage for cpp.	bl82-03209
367	cpp	incorrect line number after macro call.	bl80-29112
368	cpp	incorrect test on too many -i options	bl81-21401
369	cpp	macros arguments on two lines	bl81-27106
370	cpp/readme	the cpp/readme had a name reference that had to be scrubbed.	bl82-29401
371	cprs	speed up cprs	bl81-24702
372	cpu	instruction causes machine to mrf	mh82-15403
373	cras	perfonmance improvements and new opcodes and microcode for cras	bl82-15809
374	crash	crash 'file' option does not work.	bl82-18114
375	crash	crash does not compile in 5.0	bl82-12010
376	crash	crash does not print text symbols in overlay sections correctly.	ua82-02201
377	crash	crash does not recognize 5.0 sgs header format or namelist structure.	ua82-06102
378	crash	crash doesnt work right in 5.0.	mh82-20702
379	crash	crash used with the i option not working correctly	bl82-15405
380	crash	crash uses /usr/include/sgs headers.	bl82-13308
381	crash	crash.mk	bl82-13105
382	crash	crash.mk has to take out dependancies on include/sgs.	bl82-16007
383	crash	ctrl-d should not give an error message to crash	bl81-13406
384	crash	debugging print lef in delivered code	bl82-00815
385	crash	several problems associated with the crash program	bl81-28708
386	crash	stack dump prints wrong addresses	bl82-00814
387	crash debugging	crash debugging enhancements	bl81-29611
388	crash(1m)	crash(1m) mishandles some of "proc -" on pdp 11/70	bl81-05011
389	crash(1m)	debug statements left in	bl81-09401
390	crash(1m)	document invalid tty structure	bl81-09403

NR	AREA	SUMMARY	MR
		message	
391	cronlog	relocate cronlog to /usr/adm	bl80-28111
392	crypt	problem in decrypting multiple files	bl81-13405
393	crypt.1	need warning about concatenating files	bl81-22224
394	csi	need changes to csi driver for csi conversion.	bl81-31504
395	csi	the common syncronous interface (csi) should be included in release 4.2	bl81-25803
396	csplit	csplit fails with illegal offset when arg1 is '/rexp/-1'.	bl80-27701
397	ct	add useful ct enhancement	bl81-23701
398	ct	ct doesn't hang up line on termination	bl81-00903
399	ct	ct fails to emit login and then hogs acu line.	cc82-05502
400	ct	ct will not run with cbinit.	bl82-17533
401	ct	update to current version for unix 4.0.	bl80-28212
402	ctime	ctime's "cbuf" should be static.	bl80-00308
403	cu	cu command option documentation	mb81-24017
404	cu	cu command ~nostop doesn't work	bl81-01314
405	cu	cu does not allow you to toggle the nostop option.	bl80-29732
406	cu	cu gets tangled up during file transmission	bl81-05022
407	cu	cu hangs and takes over the cpu	bl82-20209
408	cu	cu returns too early.	bl81-03501
409	cu	cu sets clocal on direct lines	bl81-06208
410	cu	cu should recognize ~ after all line delim.	bl81-06501
411	cu	manual page for cu documents -a option, has error on nostop feature.	bl81-08506
412	cu	on line disconnect cu transmit routine still leaves shells running	cc81-35001
413	cu	please enhance cu	bl80-26505
414	cu	the cu manual page has a typo	bl80-24801
415	cu	typing quit kills cu	bl80-32051
416	cu	~$ is not documented on cu manual page	we81-05102
417	cu command	cu command eats up cpu if not terminated gracefully.	bl82-02512
418	cu command	erase character handling on escaped lines	bl81-04106
419	cu command	the cu program aborts if very long lines are entered.	bl81-01406

NR	AREA	SUMMARY	MR
420	cw	cw not portable	bl80-36508
421	cxref	cxref doesn't do seperate cross-references for multiple files	ll81-07001
422	cxref	fix references to <sys/signal.h	bl81-02710
423	cxref	install cxref (new cross-reference program).	bl80-32052
424	cxref	smallish (pdp11 size) tables limit program domain	bl82-04108
425	cxref	sub-components misplaced	bl80-32910
426	cxref(1)	-c option doesn't work as advertised	bl81-08003
427	czabc	czabe does not check bwqac for x'82'	bl80-31227
428	date(1)	misuse of word julian.	bl80-35307
429	date-ab	should be changed to conform to new software from dept. 3644	bl80-21209
430	date.c	incorrect data declaration for argv in main(argc, argv)	bl81-27403
431	dc	dc loops	ua81-35603
432	dc	too may decimal points confuse dc	bl81-12502
433	dc cmd	exponentiation problem	ua81-32401
434	dcopy	dcopy options not documented.	we81-12413
435	dcopy	documentation discrepancy.	bl81-24306
436	dcopy(1m)	dcopy needs separated i/d	bl81-13703
437	dd	changes needed for dd to work under unix-rt.	bl81-14610
438	dd	dd conv=swab fails (sometimes) to swab last bytes	bl80-22422
439	delivery e	missing parts for delivery e.	bl82-09506
440	delta	-m flag accepts a bad format	bl80-28806
441	delta	automatic null deltas screw up sccs file	bl80-32017
442	delta	delta corrupts s file	bl80-31603
443	delta	delta gives the wrong error codes when inex conflicts occur.	bl80-33606
444	delta	delta without get drops core	bl80-35112
445	delta	limit on change commentary is much too small.	bl81-08004
446	delta	on the 11/70 if there are many deltas in a file delta says out of space	bl81-29602
447	delta command	the delta command does not close files when an error is found.	bl81-20207
448	deltack	calls to msg have wrong number of args.	bl82-02803
449	deltak.c	pfile undefined in deltak.c	bl82-12007
450	deroff	deroff inserts characters	bl79-23307
451	descend (subroutine)	descend is a function which descends a directory	bl80-29716
452	devnm(1m)	devnm results unclear	bl80-29009

NR	AREA	SUMMARY	MR
453	devnm(1m)	names not recognized should be error	bl80-29008
454	df	df incorrectly tests for invalid file systems on pdp11	bl82-12404
455	df	df reports that filesystems are not filesystems	bl82-17801
456	df	options cannot be used together	bl81-23904
457	dfc	a failed i/o operation does not free dma jobid for reuse.	bl82-25605
458	dfc	dfc changes for generic 2 microcode	bl82-16904
459	dfc	dfc restart during diagnostics hangs dfc.	bl82-21605
460	dfc	driver sets wrong bits for defect table initialization for 675mb disks.	bl82-27104
461	dfc	dskvfy should give details about tracks found to have problems.	mh82-23501
462	dfc	more messages needed to diagnose dfc/disk problems	bl82-27811
463	dfc	multiple timeouts running to correct dfc hang after power failure	bl81-30156
464	dfc	need a software work around for dfc firmware deficiency.	bl82-23401
465	dfc	need driver help for firmware problems with bad 675mb defect tables.	bl82-27805
466	dfc	need software to rewrite defect tables to bypass firmware bugs.	bl82-29505
467	dfc	need to make changes to fix bugs detected during unit testing	bl82-21022
468	dfc	the adskvfy command requests the wrong number of tracks.	bl82-30503
469	dfc	the dfc must be changed to support system activity reporting	bl81-25804
470	dfsck	changes needed for dfsck to work under unix/rt.	bl81-13407
471	dfsck	dfsck should correctly interpret '-t' option arg on cmd. line.	bl81-05012
472	dfsck	handling of pipe errors.	bl81-05014
473	dfsck	source code clean-up.	bl81-13408
474	diagnostics	diagnostics sometimes crash unix	bl81-26417
475	diagnostics	iop/dfc diagnostics	bl81-25907
476	diagnostics	the mt diag. test tape should come with the system	bl82-07713
477	diff	diff should have separate text and data areas.	bl80-11221
478	diffmk(1)	remove personal pronoun from manual entry	bl81-31316

NR	AREA	SUMMARY	MR
479	diffmk/tbl	diffmark does not work with tbl.	bl80-09908
480	dir(5)	dir(5) writeup needs <sys/types.h	bl80-29403
481	dircmp	dircmp can't handle files with embedded blanks	bl80-32054
482	dircmp	manual page is incorrect.	bl80-25101
483	dis	change order of includes in dis_main.c	cc82-03401
484	dis	changes to dis for 5.0 simplex release	uz82-05602
485	dis	changes to dis to be consistent with new ar.h	uz82-05701
486	dis	fix floating point operand disassembly	bl81-25820
487	dis	incorrect spec for string instructions "movcce" and cmpcce"	bl82-06407
488	disk driver	under fault cond disk drivers unable to handle certain faults during some peripherial ops.	bl81-34312
489	dj(4)	remove dj(4) man page from manual	bl80-27614
490	dl	nosdp ld doesn't handle dot assignment stat correctly	bl81-16304
491	dmc	the dmc driver does not have an ioctl interface similar to the dmr and dmb drivers.	bl81-03726
492	dmc dmr	the dmc and dmr drivers incorectly frees the unibus map registers, causing the system to c	bl81-03725
493	dmc drivers	the dmc driters should allow the maintence mode bit to be set.	bl81-24505
494	dmc drivers	the dmc drivers should allow the maintance mode bit to be set.	bl81-24504
495	dmc(4)	cannot bring machine down cleanly with 3.0 dmc driver	bl81-01401
496	dmc11-fa driver	addition of ioctl to dmc driver.	bl81-22220
497	dn driver	add delay for shared acu operation	ua81-32302
498	doc	change to sh manpage.	bl82-03211
499	doc. for ipc	the manual does not mention the need to include the	bl81-21513
500	docs	error on checking root	bl81-30138
501	docs	let's get machine-readable docs into 4.2	bl81-25101
502	documentation for acct(5)	wrong claim in acct(5) for mean process size	bl82-07705
503	documentation of error msgs	should be better doc. of unix error msgs	bl82-01327
504	doprnt.s	c preprocessing should be replaced by m4 preprocessing.	bl82-07108
505	doprnt.s	fails on unexpected "l" in printf format.	ua81-35601
506	dpd	partial jobs are being executed on	bl82-03602

NR	AREA	SUMMARY	MR
		gcos	
507	dpd.c	doesn't return errno on interrupt.	bl81-13402
508	dpr	cannot communicate with the honeywell printer.	bl82-18109
509	dpr	local changes to dpr	bl80-34017
510	dpr	unrecognized option	cc82-00502
511	drand48()	drand48() does not work on pdp11.og=st	bl82-01328
512	dscbs documentation	dscbs documentation is in error	bl80-31257
513	dskfmt	dskfmt leaves mhd in diagnostic state	up82-05301
514	dst.c	doesn't clear d-wait on interrupt.	bl81-27201
515	du	du does not know about "null" file-entry names.	bl81-05016
516	du	du does not work properly with new 1k file systems	bl82-19502
517	du	erroneous output if more than 8 open directories.	bl81-02703
518	du(1)	du does not work with directories with greater than 64 entries	bl81-02909
519	dump	dump clears channel before clearing dma.	bl82-29506
520	dump	dump incorrectly thinks .tv is static	bl81-24028
521	dump	dump options don't work: -tv gives ?'s, -t not parsed correctly, -d documented poorly, -z,	bl82-06205
522	dump	dump should retry	bl81-32105
523	dump	dump uses 300mb section table for 675mb disks	bl82-25907
524	dump	if dump fails, it will retry indefinitely.	bl82-01001
525	dump(1m)	documentation wrong	bl79-20004
526	dump-ht.s	can't get post mortem dump to work.	bl81-06203
527	dz/kmc driver	bug in kmc program for dz assist, parmrk option	bl81-24008
528	dz/kmc driver	improve performance of dz/kmc interface	bl81-22221
529	dzb error messages	cryptic dzb error messages printed on console.	bl80-23306
530	dzbload	load message always printed	bl82-13703
531	ecvt.c	integer conversions are not exact.	bl82-09201
532	ed	change ed so that a pwb fspec error is a warning instead of a fatal error	bl80-03806
533	ed	ed - problem with using n and l together.	bl81-29438
534	ed	ed allows a quit after an undo	bl80-32031

NR	AREA	SUMMARY	MR
535	ed	ed allows you to quit when it shouldn't.	bl80-17102
536	ed	ed dumps core when started	bl81-19708
537	ed	ed processor gives "?" when first line of file has bad tab line.	bl79-31207
538	ed	ed should write 0 length files.	bl80-33108
539	ed	fixed botch on ed man entry	bl80-29731
540	ed	fspec(v) and the -x option does not appear to work together.	bl80-12904
541	ed	illegal character error message	bl81-28926
542	ed	inconsistent use of pwb symbol	bl80-20712
543	ed	lines of up to 512 chars cannot be entered	bl78-11602
544	ed	prevent ed from scribbling files.	bl80-29531
545	ed	prompt string	bl81-25407
546	ed	the editor ed (1) command "0,$w" emits an extra junk first line.	bl81-31003
547	ed	w !file of editor is documented incorrectly	bl80-23102
548	ed (1)	partial write from ed turns on its internal change flag.	bl81-31004
549	ed man entry	red is not documented	bl80-16510
550	ed(1)	editor writes zero	bl80-35711
551	ed(1)	minor corrections to ed document in unix 3.0 manual.	bl81-00603
552	ed(1)	use with ed is not specified.	bl80-35116
553	ed/sh	cannot keep the set-uid permissions on a file using ed	bl81-33714
554	efl	efl does not work.	bl82-27950
555	efl	improve makefile	bl81-02711
556	efl	new version	bl80-27516
557	end(3)	description lacking	bl81-26901
558	end(3c)	manual page in error	ua81-31502
559	end.s	temporary stack area not large enough.	bl80-26504
560	endopen.c	lseek without checking results of open/create	bl81-01602
561	env	return code from env is non-zero.	bl81-10305
562	env	return code from env is non-zero.	bl81-33817
563	eqnchar	add new characters to eqnchar.	bl80-25420
564	erep	erep cannot run under userid sysoper0	bl80-31242
565	erpfrm and service record file	lack of documentation for erpfrm procdef	bl80-31243
566	errdead	errdead changes for 5.0 sgs compatibility.	ua82-06801
567	errdead	errdead will not compile.	bl82-12605
568	errdemon(1m)	manual page indicates errdemon(1m) is in /etc.	bl80-35204

NR	AREA	SUMMARY	MR
569	errdemon, makekey,manprog	files missing from sccs, or files.x.y	bl82-32301
570	error logger	dma, dfc error messages.	up82-00405
571	error messages	many error messages aren't described in documentation.	bl81-31609
572	error messages	the unix op guide should contain more data such as register layouts	bl81-34408
573	error numbers	the error message printed by perror for enxio should be changed.	ua82-02602
574	errpt	add rm05 reporting.	bl81-02815
575	errpt	bad pagination.	bl81-02816
576	errpt	bad parity error reporting.	bl81-02814
577	errpt	configuration information missing	we81-10502
578	errpt	detailed report output not aligned on page boundaries	we82-01227
579	errpt	errpt outputs erronous dfc errors	bl82-13213
580	errpt	output format bad when more than one limitation specified.	bl81-27104
581	errpt(1m)	errpt with options doesn't produce correct totals	ll81-11404
582	etp(8)	removed from the manual	bl82-01701
583	everything, potentially	scrub btl references from the source code comments	bl82-27945
584	exec(2) man page	extern char **environ not documented	bl80-32033
585	export tape	allow the docs as a separate subsystem	bl81-05110
586	expr	expr hangs	bl80-20421
587	expr	expr is missing external declarations	bl82-07008
588	expr	expr performs 16 bit relational operations instead of 32 bit (on 11/70 only).	bl80-26703
589	expr(1)	expr will produce a core dump given certain erroneous input	bl80-32210
590	f77	allow comments in f77 "tokens" file	bl80-28713
591	f77	assignment to do-loop index variable incorrect	bl81-18207
592	f77	bad code generated	bl82-16711
593	f77	bad flag parsing	bl82-16710
594	f77	carriage control in formats doesn't work	bl81-29306
595	f77	deliver coff compatible f77 for vax	bl82-10804
596	f77	erf and erfc are missing from libf77	bl82-22105
597	f77	error in backspace function	bl81-05618
598	f77	f77 -0 produces bad code for	bl81-07010

NR	AREA	SUMMARY	MR
		"assign to go".	
599	f77	f77 can't compile "mod" operator	bl82-16707
600	f77	f77 fails in pass 2 on example	bl81-03712
601	f77	f77 generates bad sdb pseudo-ops	bl82-16706
602	f77	f77 list-directed input handles exponent incorrectly.	bl81-11708
603	f77	f77 makefile needs to be changed	bl82-18918
604	f77	f77 will not compile c files.	bl81-01616
605	f77	fix makefiles to clean created files	bl81-02708
606	f77	fix references to <sys/signal.h	bl81-02709
607	f77	generates non-aligned stack frame	bl82-22201
608	f77	generates non-aligned stack frame	bl82-30909
609	f77	how about some documentation on our f77?	bl81-29308
610	f77	incomplete cleanup	bl80-33401
611	f77	new version	bl80-27517
612	f77	no diagnostic issued for compile-time errors.	bl81-05509
613	f77	optimizer should be improved.	bl81-05606
614	f77	productivity quality fortran	bl81-14813
615	f77	recent changes to code generator broke f77	bl82-20202
616	f77	suggest change to the "stop" statement.	bl81-09806
617	f77	take out /usr/include/sgs header file dependancies.	bl82-17210
618	f77	the -i2 option is not specified.	bl80-05101
619	f77 compiler	the f77 compiler creates bad assembler code.	bl81-23709
620	f77 free fmt i/o wrong inputting neg. flt. pt numbers	free fmt i/o ng on inputting neg. numbers flt. pt in f77 vax only	bl80-28201
621	f77(libi77)	g format descriptor not standard-conforming	bl81-13807
622	f77.mk	f77.mk calls pcc.mk in the wrong directory	bl82-12306
623	f77/ld	loader options out of f77 boggle the mind & the loader.	bl81-19802
624	factor	factor needs factor.mk	bl82-29802
625	factor	users want it back	bl81-22318
626	fflush	fflush bug causes emrecv to fail.	cc82-03901
627	fflush	unbuffered output fails in update mode	we81-10414
628	file	file command loops infinitely	bl82-19301
629	file	file does not recognize cpio archives created with -c	bl81-23820
630	file	file doesn't recognize 11/70 archives	bl80-32039

NR	AREA	SUMMARY	MR
631	file	the file command updates the 'changed' time of files owned by the user.	bl81-09709
632	file and text tables	use linked lists for free chains of file and text tables.	mh82-05508
633	file command	the vax file command does not recognize 5.0 object and archive file formats	ua82-02004
634	file mode	make file mode verification check	bl81-30622
635	filesystem organization	no effective/fast way to optimize layout of a filesystem	bl81-01407
636	find	detection of bad dir names by find.	bl81-05820
637	find	find -name file does not recognize ! in file's shell syntax	bl81-28923
638	find	find will not always match a name correctly	bl82-11908
639	find	the find command incorrectly handles unix-rt special file types.	bl81-01412
640	float pt.	some floating point constants are assembled incorrectly	bl81-22613
641	flsbuf	bug in flsbuf involving _sobuf	bl81-01601
642	fopen	fopen causes errno to be set incorrectly in certain circumstances	bl80-27514
643	fopen.c	function _endopen() should be declared static.	bl81-24701
644	fork	need faster fork system call.	mh82-05505
645	fortran	f77, ratfor, asa, efl etc. are missing.	bl81-22621
646	fortran	update documentation for 5.0 release	bl82-03104
647	fprintf	output to a terminal becomes buffered after a break.	bl81-21105
648	frec	frec does not work with volcopy tapes	bl82-18703
649	freopen	freopen (3c) fails if stream named is not already open.	bl81-30205
650	frexp.c	delivered frexp.c missing include file	bl81-22708
651	frexp.s, ldexp.s	cc -p prog.c -lm; a.out gave emt trap -core dumped	bl82-21604
652	fs	cannot check or create fs larger than 131k blocks.	bl81-30006
653	fsba	modify triple indirect address part	bl82-18005
654	fscanf	the %c conversion character doesn't work properly	bl81-12517
655	fsck	4.0 fsck should not print device	bl81-02818

NR	AREA	SUMMARY	MR
		name on each line message.	
656	fsck	5.0 fsck on the 11/70 will not run standalone.	bl82-11218
657	fsck	allow to build free list with larger blk/cyl. value.	bl81-30401
658	fsck	another precautionary measure on fsck.	bl81-02820
659	fsck	avoid bad entries in lost+found.	bl81-25207
660	fsck	change option name.	bl81-04105
661	fsck	change to fsck man page to correspond to the source.	bl81-25408
662	fsck	changes related to 5.0 file system.	ua82-08901
663	fsck	correction to rt code and "bad-blk-in-directory" check code.	bl81-11303
664	fsck	double reconnection of directories with link count 0.	bl82-12403
665	fsck	elimination of option.	bl81-25310
666	fsck	fsck dies with no message when a directory exceeds 64 levels.	bl81-29625
667	fsck	fsck does not handle unreferenced i-nodes with a link count of zero	bl81-19804
668	fsck	fsck leaves file system corrupted.	bl82-09404
669	fsck	fsck makes bad directory entries	bl81-27107
670	fsck	fsck obliterates file systems	bl81-03017
671	fsck	fsck puts nulls to stdout	mb82-08627
672	fsck	fsck should report about bad blocks on directories.	bl81-05515
673	fsck	fsck sometimes hangs system	we81-31901
674	fsck	further capabilities of '-v' option.	bl81-02821
675	fsck	further fsck reporting.	bl81-01509
676	fsck	handling of unreferenced nonempty pipes.	bl81-28119
677	fsck	improvements to the -d option.	bl81-25309
678	fsck	increase robustness of fsck argument scan.	bl81-03004
679	fsck	integration of fsck versions.	bl81-25206
680	fsck	less-verbose fsck.	bl81-02008
681	fsck	more precautions.	bl81-02819
682	fsck	pipe removal	ua82-07402
683	fsck	provide 'fast' option to 5.0 fsck.	ua82-01801
684	fsck	report incompatible options.	bl81-28120
685	fsck	response should be checked for validitity	bl81-30107
686	fsck	signal handling by fsck.	bl81-06902
687	fsck	yet another option for fsck.	bl81-01511
688	fsdb	block assignment in fsdb is not working correctly.	bl80-32510
689	fsdb	increasing or decreasing the value	bl80-30802

NR	AREA	SUMMARY	MR
		of the size field of an inode of a large file does not	
690	fsdb	truncation check is invalid for data block addresses!	bl81-01001
691	fsdb(1)	the fsdb(1) command will not correctly change the size field.	bl80-35114
692	fsplit(1)	fsplit(1) man page is incomplete	bl81-08005
693	fts	field test set support needed	bl81-25819
694	ftw	iread is undefined	bl81-31531
695	ftw function	a program using ftw() is told iread is undefined.	bl81-32302
696	fuser	fuser won't compile	bl82-13214
697	fuser	install fuser program	bl80-32220
698	games	/usr/games/turnoff (and turnon) don't provide for setuid games.	bl81-20907
699	games	alter games directory	bl81-30001
700	games	games will not build due to bad sccs id	bl82-23508
701	games	include rogue in games	bl81-30014
702	games	mis-spelling of a word in quiz.	bl81-04332
703	games	no source or documentation for many games	bl80-35312
704	games	unavailable games set up wrong.	bl81-21003
705	games login	games logins are treacherous	bl81-27401
706	gamma() function	gamma(-1) gave errno=0 instead of 33.	bl81-23726
707	gamma(3)	gamma(0.0) gave a memory fault, core dump.	we81-22648
708	gas	gas doesn't work as advertised	bl81-25311
709	gcat	doesn't accept -s4 option	bl81-35605
710	gcos	update kmc1 protocol programs for latest vpmc.	bl81-01510
711	gcos/realconfig.c	mulitlpy-defined external variable	bl82-07804
712	gcosmail	can not end with a period.	bl81-04333
713	ged	ged core dumps	bl81-04404
714	ged	ged read command cannot read text	bl82-22107
715	ged	ged zoom (in) function locks tektronic in raw mode.	bl80-18320
716	ged	zoom causes ged to quit without saving changes	ms81-20210
717	gen/*.s	label names should start with period.	bl82-10506
718	general disk driver	generalize disk driver for other kinds of drives	bl81-04007
719	generic 2 dfc firmware	unable to boot off secondary disk	bl82-24601
720	get	an l-file can not be created when a g-file is not created.	bl80-31503
721	get	get -e to unwriteable file	cc81-22703

NR	AREA	SUMMARY	MR
		produces p file	
722	get	get -g -lp does not work.	bl80-33607
723	get	get link edited without -i flag; no space (ut9)	bl82-00702
724	get	unexpected and unreasonable results when a release is missing.	bl80-00311
725	get	use new capablities of getuid to eliminate fork in get.c	cc81-17503
726	get/delta	get and delta should uuse the real group id.	bl81-00503
727	getenv	documentation is misleading	at81-32406
728	getenv(3c)		bl81-13111
729	getenv.c	getenv should check for a null pointer	bl80-32020
730	getgid	getgid returns the uid	bl82-19612
731	getopt	getopt dumps core with wrong number of arguments	bl81-01309
732	getpass.c	getpass.c has obscure exposure.	bl80-29402
733	getpwnam(), et al	getpwnam does not work correctly with login id's	ua82-01501
734	getty	getty doesn't set xcase mode bit for login in upper case.	mh82-15201
735	getty	power failure shuts down phone lines	ms81-16302
736	getty	too many login:'s	bl80-26708
737	graphcis	multiply defined symbols in arcfill.c, softarc.c,directory /usr/src/cmd/graf/src/dev.d/upl	bl81-19505
738	graphical editor	eratic behavior while in graphical input mode.	bl81-21612
739	graphical utilities	undefined function "linemode" in /usr/src/cmd/graf/src/gutil.d/gtop .c	bl81-19506
740	graphics	"gtop1pd" produces invalid output.	bl80-21411
741	graphics	2/3 pie from gtop to ptog to td	bl82-00615
742	graphics	avoid c bug mr bl81-25703.	bl81-26606
743	graphics	avoid c bug mr bl81-25715.	bl81-26604
744	graphics	avoid c bug mr bl81-25716.	bl81-26605
745	graphics	avoid c bug mr bl81-26026.	bl81-26607
746	graphics	avoid c bug, mr bl81-25706	bl81-26603
747	graphics	axis values need truncation	bl81-26411
748	graphics	bufsiz redefined	bl82-09502
749	graphics	d in vtoc option string	bl82-02207
750	graphics	eliminate fix for mrbl81-26410	bl81-31715
751	graphics	extraneous characters written to gps file.	bl81-19407
752	graphics	gd, td, ... do not recognize eof	bl82-13121
753	graphics	ged pie output with fill	bl81-30166
754	graphics	ged tty problem	bl82-09401
755	graphics	ged zoom	bl81-25411

NR	AREA	SUMMARY	MR
756	graphics	graphics commands are missing.	bl81-22618
757	graphics	in ged, set output is not vertical.	bl82-07005
758	graphics	include same file twice.	bl81-23718
759	graphics	invisable lines	bl82-03212
760	graphics	label command in infinite loop	bl82-06007
761	graphics	label command misaligns y axis labels.	bl81-23712
762	graphics	multiple definitions for "cw" when kliknking routines from /usr/src/cmd/graf/src/glib.d/gp	bl81-19509
763	graphics	pie -pp file	bl82-05408
764	graphics	pie filled	bl81-31714
765	graphics	pie output with filled sectors into gtop	bl81-28402
766	graphics	problems creating plots on 3.0	bl80-31607
767	graphics	problems in bucket, bel, ged, gas	bl81-08213
768	graphics	reference stat memo	bl81-28406
769	graphics	remcom does not exit(0)	bl81-29219
770	graphics	sh ./cca or ./cco	bl81-30304
771	graphics	td partially displays a gps file on a tek 4014	bl82-00408
772	graphics	the graphical utilities pd and ptog fail on the vax	bl80-22003
773	graphics	ttoc	bl81-25410
774	graphics	ttoc doesn't work	bl82-02505
775	graphics	union adrbits in file xytekl.c must be inverted for unix 370	bl81-21610
776	graphics	use of "-c" is not clear.	bl81-21709
777	graphics	vtoc command won't work	bl82-06008
778	graphics	vtoc dumps core if last field on line 1 is missing	bl82-13215
779	graphics	vtoc longfilename dumps core	bl82-07004
780	graphics	whatis.d/quit	bl81-32308
781	graphics	y label of graphs	bl81-26410
782	graphics header files	fields must be reversed for 370 in /usr/src/cmd/graf/include/gpl.h	bl81-19408
783	graphics library	buffer size in /usr/cmd/graf/src/dev.d/lib/tele.h inconsist.	bl81-19510
784	graphics make file	incomplete macro assignments in /usr/src/cmd/graf/src/makefile	bl81-22503
785	graphics(pie)	problems with pie and titles.	ms81-35509
786	grep	an #ifdef should be around u370 code	bl81-28302
787	grep	fails after the first file it cannot open	bl80-35008
788	grrpt	limitation's line should have one less tab.	bl81-23916
789	gsort(3c)	the manual page claims that the	bl80-36504

NR	AREA	SUMMARY	MR
		second arg of qsort is integer.	
790	gtdump.c	gtdump should allow you to dump to other than drive 0	bl82-00804
791	header files	alter comments in filehdr.h	bl81-35223
792	header files	buy back header files	bl81-34104
793	header files	erange has a different value depending on the header file.	bl80-22425
794	header files	fix reloc.h	bl81-29659
795	help	have flag to be able to put help files in other locations	cc81-10601
796	help	help (1) dumps core on long arguments.	bl81-09211
797	help	help gives incorrect info	bl80-29005
798	help	sccs help files updated.	bl80-28107
799	hp	the hp filter doesn't work as published.	bl81-04351
800	hp.c	disk driver will not handle dual ported disk drives on 11/70's.	bl81-01614
801	hp.c	handle multiple physio requests.	bl80-17508
802	hp.c	need to be able to handle multiple controllers.	bl80-29518
803	hp2boot	unix systems larger than 133k will not boot	bl81-29509
804	hpic	re-issue of bl80-29518	bl80-32908
805	i/o error messages	all i/o error messages should specify drive number	bl81-04348
806	iltd	change generic system.	bl81-06702
807	iltd	see below.	bl81-07305
808	iltd.mk	larger boot programs needed.	bl81-07306
809	include file	change sgs.h for release	bl81-24030
810	init	init does not exec su correctly in single user mode	bl82-29301
811	init	the init process can be killed	bl81-23923
812	init	the init process can be killed	bl81-24022
813	install	mandatory nature of -o option is not documented.	ua82-08201
814	installation	need information on ppf & trace installation.	bl80-31703
815	internal	clarity internal memory allocation	bl81-22223
816	internal	clarity internal memory allocation	bl81-24007
817	intor(2)	intro (2) of user's manual has <error.h	bl81-01302
818	io	the un56 driver should be exclusive open, and print no diagnostics.	bl82-19407
819	io makefile	io.mk has the wrong dependencies fo shm.c	bl82-07515
820	ioctl	ioctl does not wait for output to drain	at81-31302
821	ioctl.h	include file improperly documented	bl81-05025

NR	AREA	SUMMARY	MR
822	ipc	a process waiting on a semaphore to become 0 won't always wakeup.	bl82-22317
823	ipc	add 5.0 ipc	bl81-24407
824	ipc	can't send messages of type 127.	bl81-03417
825	ipc	improved inter-process communication (ipc)	bl81-14720
826	ipc	install messages on pdp11/70	bl80-25414
827	ipc	install new shared memory on vax	bl80-25413
828	ipc	install semaphores on pdp11/70 and vax	bl80-25417
829	ipc	ipcs core dumps on the 11/70	bl82-18008
830	ipc	remove 4.0 ipc facilities	bl81-24406
831	ipc	shared memory should limit the amount of physical memory it uses.	ua81-32301
832	join(1)	join usage (join no arg) inexplicable by man join	bl79-33302
833	kernal	repeated no space on dev x/y messages	bl81-05423
834	kill	the kill command will not send signals to all possible unix-rt processes.	bl81-01410
835	killall	include new killall	bl81-09202
836	killall	killall not compatible with 5.0 sgs	bl82-11701
837	kmc driver	improve performance of kmc interface	bl81-22222
838	l64a	function doessn't behave as it should	bl81-21715
839	labelit	changes needed for labelit to work under unix/rt.	bl81-13915
840	labelit	part of labelit's output gets lost when using a slow terminal.	bl81-13916
841	labelit	punches hole in file system.	bl81-07401
842	labelit,ncheck, mkfs	for portability's sake.	bl81-08513
843	lang	provide machine language processors	bl81-19519
844	ld	.comm botched if defined	bl82-23905
845	ld	1 process ld should put symbols into symtb when parsed	bl81-16306
846	ld	avoid unnecessary function calls	bl81-32203
847	ld	buy back klein's bug fix	uz82-03901
848	ld	buy back ld	bl81-34207
849	ld	cannot guarantee alignment of fortran common	Bl82-16708
850	ld	change ld to handle the new archive format	bl81-29653
851	ld	common symbols need to be aligned on the vax	bl82-31502
852	ld	comparison between short and	bl82-06109

NR	AREA	SUMMARY	MR
		constant doesn't work	
853	ld	eliminate extra symbols	bl82-03915
854	ld	formatting differences in printf affect ld	uz82-03401
855	ld	ifdef includes in ld1.c	bl81-30210
856	ld	isarchive macro compares short to string with portar	uz82-02101
857	ld	ld -r botches fortran common	bl82-22403
858	ld	ld -r produces incorrect output	bl82-20208
859	ld	ld assumes all .text, .data, and .bss entries have aux entries	cc82-01502
860	ld	ld does not report an error when an output file exceeds the ulimit.	bl80-28805
861	ld	ld exits with non-zero status when -r is given.	bl81-04331
862	ld	ld exits with non-zero status when -r is given.	bl81-33717
863	ld	ld for f77	bl81-29655
864	ld	ld ignores the fact that bit field can be enum	bl81-17509
865	ld	ld produces wrong symbol type for object file symbols	bl80-35101
866	ld	ld runs out of file descriptors.	bl81-23302
867	ld	ld thinks file name symbols are text	bl81-00909
868	ld	ld1.c doesn't include a necessary header file	cc82-01201
869	ld	make common ld be able provide 2 different opt hdrs	bl81-23917
870	ld	make native mode output files executable	bl81-35508
871	ld	merge changes to ld	bl82-03922
872	ld	modify ld not to load symbol tables of plib's	bl81-29656
873	ld	multiple calls to 'ld -r' fail	bl81-04916
874	ld	provide msg on multiply defined external symbols	bl82-31204
875	ld	put out .i_l_dxx and .history sections for ild	bl81-32011
876	ld	set -a flag when -ild is set	bl81-35511
877	ld	the -vs option does not work.	ua82-06107
878	ld	turn on -r flag with -ild flag	bl81-35208
879	ld makefile	syntax error in makefile	uz82-05201
880	ldexp(3c)	ldexp(value,exp). what is the type of exp ? double or int?	bl81-29615
881	ldexp.c(3c)	behavior on overflow not consistent with documentation or assembly-language versions.	bl82-28601
882	ldft	ldft disk format option doesn't work on 675mb disks.	bl82-23808

NR	AREA	SUMMARY	MR
883	ldft	ldft should recognize status reports from the dfc and iop.	mh82-21702
884	ldft feature	no error correcting in ldft feature	bl82-09903
885	led.sh	see below	bl81-07709
886	lex	incorrectly typed cfree() call arguments	bl82-01406
887	lex	lex -r cannot find /usr/lib/lex/nrform	bl79-31302
888	lib2.mk	security hole.	bl81-08104
889	libc	add portable c source for some functions.	bl81-19901
890	libc	olibc.a should not be there	bl81-06704
891	libc	printing very large numbers with printf causes abort.	bl81-22617
892	libc	proposed deferment of directory naming for temporary files.	bl80-23109
893	libc(gen/frexp.c)	module will not compile	bl81-31601
894	libc(gen/modf.c)	module will not compile	bl81-31602
895	libc(nice.s)	nice(2) will always return 0.	bl80-00209
896	libc.a	libc.a has to be ordered in a different format.	bl82-21706
897	libc/freopen	freopen fails on a closed file descriptor.	ua81-31301
898	libi77	libi77.mk has to add an "ar ts" after doing "strip -r"	bl82-17209
899	libi77	severe bug and violation of standard	bl80-27515
900	libi77.mk	libi77.mk should be changed to handle common object format	bl82-06805
901	libl.a	fix compiler warning	bl81-30224
902	libld	add sgetl and sputl to libld	bl81-34308
903	libld	buy back header files	bl81-29654
904	libld	buy back libld	bl81-29650
905	libld	buy back libld	bl81-34105
906	libld	libld files include wrong header file	uz82-02002
907	libld	makefiles do not pass the arch variable to libld	bl82-23610
908	libm	error handling enhancement to c library	bl82-00611
909	libm	libm.mk should order the library	bl82-08103
910	libm	math library must be ordered on the pdp11	bl82-08107
911	libm	the pow routine doesnt work correctly	bl82-07604
912	libm.rc	math lib rc file does not work	bl81-22705
913	libplot(vt0 (open.c))	fix compiler warning message	bl81-30225

NR	AREA	SUMMARY	MR
914	libpw	cras blows up fdfopen since fdfopen doesn't initialize buffer pointers.	bl82-20313
915	libpw	xalloc think the address space size is 65 k on the vax	bl80-10111
916	libpw.a	correct external variable definition to static	mh82-04201
917	libpw.mk	libpw.mk needs to be fixed for common object format.	bl82-06803
918	libraries	library performance needs to be improved	bl81-21605
919	library makefiles	the c library makefile must be changed to be machine-independent.	bl82-04713
920	library profile	library profile does not work on pdp.	bl82-15304
921	liby.rc	liby.rc has the sflags variable wrong	bl82-15802
922	line disciplines	line disciplines, virtual terminals for pdp11	bl81-04006
923	line printer demon	need to abort line printer requests	bl81-05405
924	line printer driver and 4014 command	default indentation should not be part of driver	bl80-26902
925	link editor	3.0 version stamping should be compatible with cb	bl80-23817
926	lint	incorrect complaints from lint when taking address of void function	bl81-06001
927	lint	lint complains about casting struct pointers into (char *).	bl81-00906
928	lint	lint complains about portable character comparisons.	bl81-33813
929	lint	lint complains about return values	bl80-09207
930	lint	lint errors on a comment in a define.	bl81-29629
931	lint	lint fails to complain about functions used but not defined.	bl80-26020
932	lint	lint gets syntax error on valid 'c' statment.	bl81-31703
933	lint	lint is confused by external static variables	we80-26705
934	lint	lint is too big for the 70	bl82-12008
935	lint	lint should have access to definitions of the math library.	bl81-09209
936	lint	lint should warn about possible sign extension fo bit fields.	bl80-22005
937	lint	lint's standard function declaractions are incorrect.	bl81-04330
938	lint	lint: -c option not supported	bl81-22403

NR	AREA	SUMMARY	MR
939	lint	lpass2.c has the variable errmesg undefined	bl82-15803
940	lint	smallish (pdp11 size) tables limit program domain	bl82-04109
941	lint	the lint libraries use the void data type improperly.	bl81-05418
942	lint	upgrade lint to release pccr1.1 of the portable compiler.	bl81-23101
943	lint	wrong entry in lint library for stime(2) system call.	bl81-05407
944	lint / qsort man page	in disagreement	bl80-36605
945	lint library	lint should not complain about unused arguments in a lint library.	bl80-22517
946	list	a variable declaration was ifdefed out by mistake	bl82-26503
947	list	clean up and ifdef bug fix for common list	bl81-23710
948	list	list declares setbuf() as extern int but is now void()	bl82-06702
949	lockit.c	lockit does not override umask when making the lock file	bl80-35018
950	log(3)	precision is off for log() function, good only up to 6 significant figures.	we81-22645
951	login	bad uid defaults to root	bl80-00309
952	login	login changes for 4.0	bl80-32026
953	login name length	we would like logins	bl80-34506
954	login(1)	login allows root to dial in	bl80-29532
955	logname(1)	change source to not use libpw	bl81-32006
956	logs	change "o" prefix to "old"	bl81-09203
957	lorder	allow lorder to handle archives	bl81-19803
958	lp	add lp job information on trailing pages as well as cover page.	bl81-16811
959	lp	lp interface to printronix doesn't detect printer faults	bl81-20409
960	lp	lp needs to be seen and not heard	bl81-21911
961	lp	make file chamnges needed.	bl82-08805
962	lp command	'lp' command prints wrong file	bl81-31306
963	lp spooling system	outputq file not created	bl81-13702
964	lp system	lp does not properly deal with the error log file	bl81-21910
965	lpr	need multiple lpr's.	bl80-35407
966	lpr/banner.c	banner will not compile under lpr	bl82-13120
967	ls	-b option to ls is no-op	bl81-20923
968	ls	a static buffer is overflowed in ls.	bl81-21005

NR	AREA	SUMMARY	MR
969	ls	flagging directory names in 'ls'.	bl80-26901
970	ls	ls does not work correctly when -o and -g options used together.	bl81-00702
971	ls	ls returns 0 even if a directory it tried to read was logically unreadable	bl80-03603
972	ls	potential bug in ls	bl81-23004
973	ls.c	why is signal.h #included?	bl81-16807
974	lsearch	does not maintain a table of structures of arbitrary elements	bl81-06110
975	lsearch	lsearch mishandles table entries with embedded null's	bl81-07610
976	lsearch.c	lsearch() library function faulty.	ua82-02804
977	m32ld	m32a.out(s) .text starts at 0x800, 000	bl81-26112
978	m4	incomplete cleanup	bl80-34514
979	m4	m4 doesn't compile in 5.0	bl82-12013
980	machdep.c	change to initial printf.	bl80-32911
981	macro	ns get wrong list when over 2 pages.	bl80-26612
982	macros.d/macros.mk	nroff -cm core dumps with illegal instruction	uz82-07201
983	mail	"to" line in bulk mailings.	bl80-21102
984	mail	adding timezone to mail timestamps	bl80-30908
985	mail	error message output	bl81-27412
986	mail	fseek no-op	bl80-35806
987	mail	mail does not append to default mbox	we81-12120
988	mail	mail does not get printed	bl80-23815
989	mail	mail failing on remote cpu is not identified.	bl81-07802
990	mail	mail mis-handles signals	bl80-32037
991	mail	mail misbehaves when /tmp is full	bl80-32030
992	mail	mail received from a remote system is not forwarded.	bl81-04323
993	mail	mail refuses what unix permits	bl81-35224
994	mail	mail should create a dead.letter somewhere	bl81-02402
995	mail	mail sometimes fails to be forwarded to another person.	bl82-12011
996	mail	needs more security.	bl80-27605
997	mail	reading large mailboxes trashes dataspace	bl81-02111
998	mail	security violated by files in /tmp	bl80-23908
999	mail	the -e option is not documented.	bl80-26015
1000	mail	the mail command hangs when parsing options	ua82-13802
1001	mail	undocumented option in mail.	bl81-10201
1002	mail(1)	need crc on mail	bl80-19205
1003	main.c	maxmem should be an int.	bl81-03308

NR	AREA	SUMMARY	MR
1004	mainst	operator' screen did not return after inputting mode command	bl80-31239
1005	make	$$ parameter not working; make -d gives carriage returns instead of newlines	bl80-20415
1006	make	$$/ does not work as advertised	bl80-31801
1007	make	doesn't understand archives	bl81-02003
1008	make	make -t doesn't work	bl80-26620
1009	make	make -t option does not work: no dates get touched.	bl80-35022
1010	make	make does not work on 5.0 archives	bl82-09009
1011	make	make doesn't always say "up to date" when it should	bl80-32218
1012	make	make shouldn't have separate pwb rules.	bl80-20711
1013	make	manual entry error	bl81-09305
1014	make	problems with make	bl80-03901
1015	make	problems with make on 11/70	cc81-16902
1016	make	remove obnoxious comment about murray hill and unix	bl82-28602
1017	make	touch option in make fails.	bl80-28002
1018	make (1)	.precious usage explanation is confused	bl80-24509
1019	make (1)	comments in shell commands mishandled	bl80-24521
1020	make (1)	comments on macro definition lines cause problems	bl80-24526
1021	make (1)	confused explanation of use of silent and ignore characters.	bl80-24507
1022	make (1)	failure to use cflags consistently in internal rules	bl80-24505
1023	make (1)	filenames can not begin with the '@' character.	bl80-24519
1024	make (1)	inconsistent blank handling in multiple line shell commands	bl80-24517
1025	make (1)	need for blanks in multi-line shell commands	bl80-24506
1026	make (1)	spaces in library dependents cause anomalies.	bl80-24501
1027	make files	make for sccs libraries fails due to problem with strip command	cc81-27401
1028	make files	ownership and group id of testjob1 and testjob2 should be made consistent	bl82-06303
1029	make(1)	make does not understand 5.0 object file and archive file formats.	bl82-04808
1030	makefiles	"opr", an obsolete command, must be removed from makefiles.	bl82-30703
1031	makefiles	add -dportar to makefiles	uz82-05308

NR	AREA	SUMMARY	MR
1032	makefiles	security holes.	bl81-07710
1033	malloc	malloc returns bad pointers under certain condition	bl80-34702
1034	malloc man page	calloc synopsis wrong	bl80-28210
1035	man	missing "see also" entries.	bl81-06306
1036	man command	bug fixes to man command	bl80-21012
1037	man macros	remove man macros for version 6	bl80-35315
1038	man pages	fix incorrect files	bl81-29646
1039	man(1)	new manual directory structure requires new man cmd	bl81-21912
1040	mancvt	mancvt(1) is referenced but not found	bl80-35311
1041	manpages	manpages needed for mlp common object file format software.	bl82-03510
1042	manprog	should be documented	bl80-20709
1043	manual	combine mm and mmchek manual entries.	bl80-16508
1044	manual	combine sh and rsh entries.	bl80-16509
1045	manual	compatability changes needed in section 2	bl81-25308
1046	manual	disk devices should list magic numbers	bl80-27613
1047	manual	need new procedure.	bl80-17610
1048	manual - sectin 3	man-pagepfor tsearch(3c) is missing	bl81-04001
1049	manual page macros	change release string to 5.0	bl82-07720
1050	manual page macros	update manual page macros for 4.0	bl80-35118
1051	manual pages	convert manual pages	bl80-02301
1052	manual pages	lpadmin, lpmove, lpsched, lpshut should be in a_man	bl81-33412
1053	manual pages	processor-types commands undocumented	bl81-28702
1054	manual sect 3	which libc routines use stdio?	bl81-06308
1055	manual section 3	the section 3 manual pages need to be updated to reflect 5.0 release.	bl82-03105
1056	master	dr11b not in master file	bl80-32001
1057	master	include "disk" entry	bl81-07303
1058	math library	fmod returns negative numbers	bl81-05617
1059	math library	id keywords are missing from six math library functions	bl82-07505
1060	math library build	the math library build shell script should be replaced with a makefile.	bl82-02911
1061	maus	erroneous header file description	bl81-05706
1062	maus	fix bugs in maus	bl81-04317
1063	mdcc	mdcc compiler renamed to mcc	bl82-23209
1064	memory management	appears to be no real memory available to system.	bl81-30004

NR	AREA	SUMMARY	MR
1065	microcode	jsb and cmp negative word instructions fail	bl82-23113
1066	microcode	need cras microcode for 5.0	mh82-15401
1067	microcode	need to fix cmpcc instruction.	bl82-21103
1068	mistake in signal_.c in f77 library	mistake in signal.c code in f77 library	bl81-23814
1069	mkdir	changes needed for 4.0 mkdir to run under unix-rt.	bl81-14611
1070	mkdir(1) man page	mkdir man page gives incorrect info on mode of directory created	bl80-23503
1071	mkfs	additional table information.	bl81-08210
1072	mkfs	defaults for gap and cylinder size should be compatible with fsck	bl81-30628
1073	mkfs	fix to bl81-01933 has been modified.	bl81-18803
1074	mkfs	largest file system mkfs can create and fsck can check is 131000 blocks	at81-35119
1075	mkfs	mkfs allows a filesystem to be made on mounted filesystems.	bl81-01933
1076	mkfs	mkfs did not work on pdp11	bl82-13401
1077	mkfs	mkfs makes filesystems that have not . or .. entries.	bl82-14408
1078	mkfs	mkfs should have more sensible defaults for gap and blocks	bl81-25817
1079	mkfs	mkfs should provide more blocks per cylinder as default.	bl80-30807
1080	mkfs(1m)	clarify default gaps and blks/cyl	bl81-08215
1081	mkfs(1m)	esystems.	bl82-13812
1082	mkfs(1m)	maximum configurable inodes not documented	bl81-20804
1083	mknod	/etc/mknod should print diagnostic message	bl80-25416
1084	mknod	the mknod command will not create unix-rt special file types.	bl81-01411
1085	mknod(1m)	incorrect synopsis specification	bl81-31303
1086	mktemp	mktemp destroys strings if a bad prototype is given.	pt81-28124
1087	mktemp	mktemp destroys the null at the end of a string with no x's.	bl81-20902
1088	mlp	the mlp software doesn't have common object file format.	bl82-03509
1089	mm	-x option of mm commands is probably unnecessary.	bl81-06202
1090	mm	bad terminal name.	bl80-12702
1091	mm	define two macros for constant-width font.	bl80-28813
1092	mm	footnote in .tl appears on abstract but not on memoranda.	bl80-27511

NR	AREA	SUMMARY	MR
1093	mm	inh is a new location code.	bl81-00509
1094	mm	location codes inconsistent.	bl80-26508
1095	mm	missing cut marks	bl80-22819
1096	mm	mm confused by display type switches.	bl80-24106
1097	mm	mm writes diagnostics on stdout.	bl80-07709
1098	mm	naming conventions not respected.	bl80-32042
1099	mm	need control over ref. list spacing.	bl80-32612
1100	mm	need to fix br and sm	bl81-22421
1101	mm	new location codes	bl80-32511
1102	mm	problems with mm	bl80-35115
1103	mm	reference page not listed in table of contents.	bl80-26507
1104	mm	references don't work inside footmotes.	bl80-16105
1105	mm	rp macro should allow for more control over page breaks.	bl80-33822
1106	mm macros	multi-line .tl broken	bl80-18407
1107	mm macros	put back the o argument to .as	bl80-17511
1108	mm macros	the .s macro is broken	bl80-21211
1109	mm man	undocumented error messages	bl80-32018
1110	mm manual	another missing error message	bl80-32019
1111	mm(1)	mm needs smarter error message	bl80-27518
1112	mm--memorandum macros (june 190 revision?)	miscellaneous problems with mm manual.	bl81-24802
1113	mmchek	mmchek incorrectly specifies "unfinished .bs"	bl80-16107
1114	mon.c	doesn't write mon.out	bl81-27202
1115	mon.h	time should be accumulated in unsigned shorts, not signed shorts.	bl82-08316
1116	mount	changes needed for mount to run under unix-rt.	bl81-14612
1117	mount	error reports in "mount"	bl81-01516
1118	mount	mount doesn't check arguments	bl79-28814
1119	mount	non-printing chars appear in output of /etc/mount	bl80-28714
1120	mount	usage message is incorrect	bl81-05018
1121	mount, umount	does not give descriptive error message for errno 15	bl81-03728
1122	msg veneer	old msgs veneer on vax does not allow recv of type 128	bl81-04318
1123	msgs	put in no-copy messages for vax	bl81-04104
1124	mv macro package	insufficient documentation.	bl80-24912
1125	mv macros	typo	bl81-25801
1126	mv program	enlarge block size for efficiency	bl81-03009
1127	mvt	bad "help" message	bl80-32032
1128	mvt macros	need some robustness fixes	bl81-22413

NR	AREA	SUMMARY	MR
		installed	
1129	na	rl01/02 driver	bl81-02103
1130	ncheck	dumps core	bl81-07402
1131	ncheck	ncheck -s does not work as expected.	bl82-07519
1132	ncheck	ncheck cannot read super-block	bl82-13016
1133	negative uids	many programs incorrectly print negative uids	bl80-32002
1134	net	'net' causes problems on systems with unlike /etc/passwd's	mb82-01728
1135	net	net man page must be modified	bl82-04306
1136	net	the net(1) command accept only one character for a system name.	bl82-01305
1137	newer (new cmd)	new command - find newest (oldes, equal) file.	bl80-31806
1138	newer(1)	unneeded command	bl81-02621
1139	newform	newform cannot handle more than 10 tabs in tabspec.	pt81-16713
1140	newform	newform drops core all the time	bl81-08507
1141	newform	newform receives memory fault	bl82-08807
1142	newform	opinion about coding style.	pt81-12404
1143	newform(1)	newform(1) command needs +s option	bl80-31003
1144	newform(1)	newform(1) dumps core.	bl81-10701
1145	newgrp	have newgrp execute .profile.	bl80-21410
1146	nice(1)	the nice command doesn't take "--#" anymore	bl81-35517
1147	nice(1)	the nice command is too nice	bl81-23715
1148	nl	-n[ln] should be -n[l]	mb81-22625
1149	nl	cannot start or increment line numbers by ten.	bl80-27606
1150	nl	expand delimitor capabilities	bl80-01107
1151	nl	nl - doesn't read std input via pipe or <	bl81-03010
1152	nl	number of file pointers left opened by nl.	bl80-01106
1153	nl	separation character prints on every line.	bl80-12906
1154	nl command	-n option does not agree with the documentation.	bl81-15511
1155	nlist(3)	nlist fails on large files	bl80-32221
1156	nlist(3c)	nlist fails on large a.out	bl80-32086
1157	nm	doesn't recover from bad file args properly	bl82-05405
1158	nm	garbage when archive file name is 14 characters.	bl81-21601
1159	nm	makefile installs non-existent file	bl81-26029
1160	nm	nm -u doesn't sit well with sdb	bl81-27506
1161	nm	nm address radix	bl81-20301
1162	nm	nm default format for 'value' is	ua82-06106

NR	AREA	SUMMARY	MR
		inappropriate.	
1163	nm	nm doesn't work if arguments are archive then file.o.	bl80-24602
1164	nm -s	the vax nm -s option is messed up.	bl81-18108
1165	nm command	nm does not close files when they cause an error	bl80-35021
1166	nm, size	nm, size not fit for standard unix usage	uz82-02003
1167	nm,size,strip	add new archive format	uz82-05309
1168	nm.c	included u370 along with vax dependencies.	bl81-30114
1169	nohup	returns "not found" for nonexecutable file.	bl81-04403
1170	nroff	nroff doesn't accept the 016 and 017 character sequences.	bl79-35503
1171	nroff	nroff doesn't draw en extender, can't draw boxes.	bl80-12204
1172	nroff	tab requests not aligned.	bl80-19305
1173	nroff/troff	if block bracket ignored.	bl80-11403
1174	nroff/troff	line continuation is broken	bl80-32609
1175	nroff/troff	split nroff and troff man pages apart.	bl82-02902
1176	nsl	compiler translates constant -1 incorrectly	bl81-19003
1177	o.s.	floating point register saving.	bl81-01201
1178	oas	em takes too long and break really screws up	bl81-21502
1179	od	octal dump doesn't handle decimal properly.	bl80-23403
1180	od	od does not return 0 on a successful exit	bl81-13511
1181	od	od returns a return code of 10 (not zero) on ending.	bl81-12101
1182	omkfs(1m)	omkfs should be documented in manual or unix's docs.	bl82-14703
1183	on-line diagnostics	on-line diagnostics report incorrect error messages.	bl82-06903
1184	online diagnostics	diagnostics may not work on a 1m system.	bl81-34316
1185	online diagnostics	mhd diag. ph 8 fails in 4.2 but not in 4.1.	bl82-06004
1186	operating system	creeping crud	bl81-34315
1187	operating system	file.h needs to add f_next and ffreelist.	bl82-13312
1188	operating system	i/o trap handler doesn't work with nsc	mh82-17902
1189	operating system	setuid programs interact with kill system call incorrectly	bl80-19911
1190	operating system	should use new microcode	bl82-01502
1191	operating system	sundry os bugs	bl81-30911

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
1192	operating system	system doesn't set/clear "block hw check" in ssr	mh82-07401
1193	operating system	too many #define's	mh82-17301
1194	operators guide documentation.	operator's guide to sss needs enhancements	bl80-31256
1195	opr	messages	bl80-21907
1196	optim	dfsck compiled with optimizer does not work correctly	bl82-18917
1197	optim	optim can make bad optimization	cc82-23501
1198	optim	optimizer breaks code when a bit-field value is returned	ua82-27801
1199	optim	optimizer incorrectly handles certain pairs of move instructions	cc82-23201
1200	optimizer		bl82-06309
1201	optimizer	bad live/dead analysis in pci	bl82-09302
1202	optimizer	deletes before .align	bl82-06501
1203	optimizer	deleting labels	bl82-06809
1204	optimizer	floating point to optim	bl81-23818
1205	optimizer	jumps to rets	bl82-12401
1206	optimizer	need to mr numbers	bl82-09806
1207	optimizer	optim loops on vax	uz82-08406
1208	optimizer	optimizer can fail with functions returning "unsigned"	cc82-21001
1209	optimizer	optimizer produces incorrect code for profiling	ua82-11201
1210	optimizer	optimizer produces incorrect code on the vax.	bl80-08104
1211	optimizer	optimizer removes string literals in text	bl81-32014
1212	optimizer	optimizer should improve jmp to ret	ua82-11205
1213	optimizer	problem with the -o option	bl82-27401
1214	os	"portable" os can't handle non 2n blocksize.	bl79-01904
1215	os	add ih comp center improvements	ua81-32101
1216	os	change internal buffer allocation on vax	bl82-07603
1217	os	changes for 5.0 sgs changes	bl82-06104
1218	os	etp memory test hangs on vax with 1 meg memory	bl81-31307
1219	os	more changes for 5.0 sgs changes	bl82-07602
1220	os	more changes for changes	bl82-08201
1221	os	non-canonical tty timeout fails if stty -brkint specified.	bl81-05409
1222	os	potential race in open/close processing	bl82-10302
1223	os	provide stand-alone driver for tu78 tape drive	bl81-26201
1224	os	setpgrp syscall bug	bl81-06120
1225	os	should be able to toggle setuid	bl80-18412

NR	AREA	SUMMARY	MR
1226	[illegible]	translation buffer not invalidated during break system call	bl82-14801
1227	os	wrong arg to sleep/wakeup in uba.c	bl82-11704
1228	os kernel	the routine timeout in clock.c can overwrite memory beyond the callout table.	bl80-34306
1229	os map tables	os map table capacity should be larger	bl81-23611
1230	os printf's	should enhance /dev/osm interface	bl81-23608
1231	os source code	changes due to 5.0 sgs	mh82-11502
1232	os source code	optimize c source code	mh82-05507
1233	os-tty	race in ttopen() - ttclose()	bl81-20402
1234	osdd	man entry needed for osdd	bl80-14104
1235	oss	losing semaphores spuratically.	bl82-29510
1236	ovld	need better one.	bl81-08313
1237	ovld.mk	there ain't one.	bl81-07301
1238	pack formatters	pack formatters should be combined.	bl81-05801
1239	pack,unpack	pack and unpack do not preserve file modes.	bl81-24414
1240	param.h/signal.h	same defines in signal.h and param.h	bl80-07708
1241	passwd	passwd aging does not allow 8 char. user ids to login	bl82-09003
1242	passwd(5)	password aging documentation should be clearer	bl79-26315
1243	pathedit (dmert)	change pathedit to use machine id's	bl82-25104
1244	pc/.c	needs power fail recovery.	bl81-11706
1245	pcat	pcat (unpack) does not work on multiple files	bl81-23003
1246	pcat	pcat fails on multiple files	bl81-07519
1247	pcat,unpack	integration of 2 separate revisions into 1.	bl80-20009
1248	pcc	guarding #ifndef for "main()" in local2.c should be changed back	bl82-10307
1249	pcc	mulitlevel option in pccr1+ causes compiler errors	bl81-22608
1250	pcc	pcc aborts on illegal construction	bl81-00708
1251	pcc	portable c compiler allows octal numbers to contain '8' and '9'	bl80-35113
1252	pcc	update pcc and fix bugs.	bl80-29534
1253	pcc	upgrade pcc to the latest release of the portable compiler and improve	bl81-23102
1254	pcc 364r1	change for side effects on (unsigned < 0) or	bl80-35207
1255	pcc 364r1	constant expression 32*2048l given incorrect shape	bl80-35303
1256	pcc code	generates incorrect code for	bl81-29609

NR	AREA	SUMMARY	MR
	generator	double args in ?: trees	
1257	pci (optimizer)	bad ld analysis	cc82-09002
1258	pcl	pcl close stores the a null ptr.	bl81-13513
1259	pcl	the pcl driver should support two pcl's.	bl80-23301
1260	pcl	waits forever for remote open.	bl81-13403
1261	pcl driver	include file in wrong directory.	bl81-03801
1262	performance	improve performance of process switch	bl81-24011
1263	performance	normalize output	bl80-29729
1264	peripheral	provide support for ml11	bl81-21803
1265	perror	diagnostics misleading	bl81-02813
1266	physio	raw into shared segment does not work with separatedi and d space.	bl81-08401
1267	plot	examples of space calls incorrect.	bl80-24720
1268	popen(3s)	documentation for popen(3s) and stdio(3s) don't agree	bl81-01310
1269	popen(3s)	documentation for popen(3s) and stdio(3s) don't agree	bl81-33715
1270	popen(3s)	documentation for popen(3s) and stdio(3s) don't agree	bl81-33815
1271	portable c library doscan.c	incorrect use of varargs.h macro	bl81-21804
1272	pow()	pow() function is only good up to 5 significant fugures.	bl81-22638
1273	pr(1)	pr(1) -n option doesn't work with multi column.	bl81-23010
1274	printf	conversion rules for printf different between 1170 and vax	we81-21504
1275	printf	printf fails at interupt level	bl81-03706
1276	printf	printf loses data after seeks	mh82-27803
1277	printf	printf no longer prints (null) in 0 pointer.	bl80-22703
1278	printf	printf on the vax	bl81-03704
1279	printf %d	printf 05%d prints leading zeroes on dec.	bl81-28103
1280	printf (vax)	printf bug in machine language version for vax.	bl81-20918
1281	printf(3s)	"%lf" does not work.	bl81-35007
1282	printf(3s9	printf mishandles %lo	bl81-01618
1283	printf/ecvt	incompatible output for "e" conversion of zero.	bl81-31309
1284	prof	negative section number may be used to index into an array.	bl82-09907
1285	prof	prof is broken.	bl80-34903
1286	prof command	one wrong symbol in "-g" output.	bl81-29601
1287	prof(1)	prof -v dosen't work	we81-22647
1288	prof(1)	prof doesn't work on the pdp11/70.	bl82-16006
1289	prof.c	unusual percent values in prof.c	bl81-31517
1290	profiler	doesn't work for text and data	bl81-02712

NR	AREA	SUMMARY	MR
1291	profiler	fix it.	bl81-08312
1292	profiler	prfpr command doesn't work	mh82-22201
1293	profiler	printing troubles	bl81-04358
1294	profiling	profiling is inefficient because of inappropriate function linkage.	bl82-11009
1295	profiling	should allow more than 500 text symbols	bl81-35225
1296	prompat.o	cc microcode does not set floating-point trap flags correctly.	bl82-23901
1297	prs	-l and -e option identical effect !!!!!	bl81-17007
1298	prs	inconsistent data format.	cc81-09706
1299	prtv utility	lib changes cause fatal error in prtv	uz82-08003
1300	ps	.profile affects ps cmd.	bl80-13004
1301	ps	adopt changes to ps command.	bl80-32308
1302	ps	faster ps available	bl81-25210
1303	ps	ps does not handle a list of uid's correctly for '-u' option	ua82-00601
1304	ps	ps must check change time of '/unix'	bl82-10902
1305	ps	ps prints userids as signed integers	bl80-26807
1306	ps	the tty field of ps command should be enlarged.	bl81-00612
1307	ptc subr.	putc return error	bl80-20505
1308	ptog command	labels do not appear in plots of files generated by ptog.	bl81-20311
1309	ptog command	plot (5) "n" command not handled correctly by ptog.	bl81-20312
1310	ptrace	in uts/vax/os/sig.c/psig() ignored signal not checked for.	bl81-04013
1311	ptrace	ptrace allows busying of text without write permission on file	bl80-33202
1312	ptrace	text of a shared-text program can be busied without file write permission	bl80-33109
1313	ptx	ptx is not complete as distributed.	bl80-10106
1314	putc	putc fix from research needs taking.	bl80-16505
1315	putchar man page	type is missing	bl80-36608
1316	pw library	curdir does not work on filesystem not mounted on root	bl81-08918
1317	pwb.mk	all entries for dqs11 should be removed	bl81-31317
1318	pwb/mm	figure caption for figure 1 - title is needed.	bl79-19905
1319	pwb/mm	numbered paragraphs needed for	bl79-19904

NR	AREA	SUMMARY	MR
		osdd.	
1320	pwb/mm	sp doesn't work with neg. numbers.	bl79-20010
1321	rand	rand function is not so random	bl81-16909
1322	rc	rc should remove uucp lock files.	bl81-03713
1323	read/write	inconsistent type for 3rd argument to read(2) and write(2).	bl81-01904
1324	realloc	realloc does not correctly calculate size of needed memory	bl80-32201
1325	reform	continue support for the reform command.	bl80-21305
1326	reform	reform does not handle input from pipes properly.	bl80-24703
1327	reform	reform should not be obsolescent.	bl80-21021
1328	reform	reform(1) command marked obsolescent	bl80-23304
1329	reform	rewrite of current software	bl79-22710
1330	regcmp	regcmp produces warning when compiled.	bl80-27509
1331	regcmp	regemp produces old-style initializations.	bl81-12401
1332	regexp.h	fix compiler warnings about redeclaration	bl81-33421
1333	rewind	rewind documented incorrectly on fseek(3s)	bl81-16810
1334	rje	if send invoked with setuid bit on issues message "wrong owner for queuer"	at81-31305
1335	rje	kmc rje lacks 'testjob'	mb81-23713
1336	rje	rje does not build with new utmp format	bl82-12410
1337	rje	rje may stop sending jobs	bl81-15311
1338	rje	rje never allows enabling of disabled reader.	bl81-15207
1339	rje	rje protocol script needs to handle funny case.	bl81-15512
1340	rje	rje redial time to short.	bl80-26002
1341	rje	rje started, but no connection made	at81-35120
1342	rje	rjedisp.c needs includes for utmp.h and sys/types.h switched.	bl82-13104
1343	rje	rjehalt needs ifdef for unix/370 compatibility.	bl81-23002
1344	rje	rjeqer fails if uid not equal to owner of queued file	bl81-30003
1345	rje	sequence errors are not handled correctly by jes 3	bl81-09301
1346	rje	size parameter in "lines" file cannot be null	at81-31402
1347	rje	the rje transmit process eats up too much time.	bl81-26412

NR	AREA	SUMMARY	MR
1348	rje	the snoop program doesn't always work.	bl81-15209
1349	rjedisp	the place field in the usr= statement is limited to 25 characters.	bl80-28401
1350	rjedisp	wrong name being associated with mail from rje.	bl81-20903
1351	rjedisp.c	multiple copies of haspdisp which are looping.	bl81-20810
1352	rjedisp.c	rjedisp.c should be modified to not send no acknowledgement msgs when its queue is full.	bl82-03303
1353	rjedisp.c	rjedisp.c should check to see if internal queue is empty before issung msgs	bl82-03901
1354	rjerecv	missing newline at end of files received from rje on jes3.	bl81-20205
1355	rjestat	return code problem	bl80-22423
1356	rjestat	rjestat -c without a command works wrong.	bl81-23005
1357	rjestat	rjestat should report queued files.	bl80-23206
1358	rjexmit	rjexmit fails due to use of cast in lseek call.	bl82-00504
1359	rl driver	does not work with rl02 drives.	bl81-07404
1360	rl01 disk driver	the mask pattern in rlio should be 01777 not 0777	bl80-28718
1361	rl01/rl02	new driver	bl81-20805
1362	rm	rm command	bl80-23203
1363	rmdel/cdc	rmdel produces bad chpost lines out to termlog.	bl82-14404
1364	rmk	sdb does not work!	bl81-15501
1365	roff.d/bldtab.c	bldtab used in text is generating garbage	bl82-13803
1366	rp03 disk driver	rp03 disk driver causes system to panic when used	bl81-04020
1367	rp07	rp07(7) and /usr/include/sys/io.h disagree	bl82-00404
1368	rp6fmt	remove from pdp11	bl81-07302
1369	rp6fmt	x option of rp6fmt not working, poor documentation.	bl80-18904
1370	sa	delivered saga.c, sagb.c did not compile	bl81-22710
1371	sadd	when sadd cannot creat the daily data file no errors are printed	ll80-35301
1372	sadp	sadp command does not exist	bl81-22404
1373	sadp	sampled i/o counts are incorrect.	bl82-14006
1374	sadp.c	usage line printed for valid parameter values. no data is produced.	bl82-17528

NR	AREA	SUMMARY	MR
1375	sag	"text-sz" plotted incorrectly	bl81-24503
1376	sag	sag -y "pread/s ; pwrit/s" fails if all values are 0	bl82-05101
1377	sar	address file wrong in sar(8)	we82-04827
1378	sar	sar averages wrong if init state changes without power fail	we81-25914
1379	sar	sar prints node name instead of system name	we81-10406
1380	sar(1)	incorrect return code from sar(1)	bl82-13208
1381	sar(1)	sar -d is not implemented	we81-22646
1382	sar.c sadc.c sa.h	upgrades are necessary for the new dfc driver.	bl82-17535
1383	sbrk	sbrk should check for zero argument	ua82-10506
1384	scanf	doscan mishandles eof in white space	bl82-05605
1385	scanf	invalid return code	bl81-06901
1386	scanf(3)	scanf code on 4.0 fails; same code on 3.0.1 is ok.	bl81-26111
1387	scanf.c	clobbers local variable.	bl81-20108
1388	sccs	%f% and %p% identical.	bl80-34613
1389	sccs	bug in sccs make file	bl80-21210
1390	sccs	delta completed but no changes recorded.	bl80-09514
1391	sccs	get runs out of file descriptors.	bl80-17605
1392	sccs	group id of sccs file being changed by non-owner user	bl80-26308
1393	sccs	improve documentation for delta	bl81-07108
1394	sccs	none of the calls to fputs() checks for error.	bl81-03314
1395	sccs	sccs makefile does not remove lib	bl81-22709
1396	sccs	sccs/date_ab.c needs dmsize and dysize declared	bl81-05201
1397	sccs	the message clock may be set wrong is confusing and unecessary	bl81-12808
1398	sccs	there is a name conflict in the use of empty in sccs and filehand	cc81-27501
1399	sccs command	cdc changes owner and group.	bl81-29432
1400	sccs(1)	sccs -r2 does not work.	bl82-14406
1401	sdb	"illegal instruction" message on interactive procedure call	bl81-27108
1402	sdb	"illegal instruction" message on interactive procedure call	bl81-28928
1403	sdb	"illegal instruction" when executing a funtion	bl81-28932
1404	sdb	a problem with sdb	bl81-27903
1405	sdb	almost all functions not working properly.	bl81-21812
1406	sdb	assorted bugs in 5.0 version of sdb	bl82-02903

NR	AREA	SUMMARY	MR
1407	sdb	botched code in vax sdb	bl81-14603
1408	sdb	cannot change location 0:	bl81-26109
1409	sdb	documentation for single stepping incorrect.	bl81-27115
1410	sdb	escapes in string print cmd	bl82-01506
1411	sdb	ever necessary bug fixes.	bl82-13107
1412	sdb	fix numerous bugs for 4.2	bl81-27102
1413	sdb	handling of unsigned and union_ed fields	bl81-27110
1414	sdb	inappropriate subprocess control on quit.	bl81-16801
1415	sdb	sdb aborts	bl82-03508
1416	sdb	sdb can show addresses incorrectly	bl81-33513
1417	sdb	sdb can't display values of pointer arrays used in c programs.	bl82-05007
1418	sdb	sdb cannot access autos	bl81-00510
1419	sdb	sdb cannot access compiler initialized structure arrays	ll81-34102
1420	sdb	sdb cannot reference unions	bl81-26113
1421	sdb	sdb does not give the correct routine contain the erroe.	bl81-33804
1422	sdb	sdb does not know about subroutine arguments in registers.	bl81-24328
1423	sdb	sdb doesn't check table indices.	bl81-13902
1424	sdb	sdb incorrectly prints register variables in 'c' format.	bl80-20704
1425	sdb	sdb lies about the values of stack variables.	bl80-20608
1426	sdb	sdb variable assignment does not work.	bl80-20610
1427	sdb	structure references in sdb	bl81-01517
1428	sdb	the sdb disassembler needs to know the new 5.0 instruction codes.	bl82-11808
1429	sdb	var!:var hangs sdb in the parsing.	bl82-01508
1430	sdb	version of sdb under sccs is preliminary, not official.	bl82-10803
1431	sdb	warning messages of files not found	bl81-27111
1432	sdb	warning messages of files not found	bl81-28931
1433	sdb	what does the directory option mean to sdb???	bl81-08102
1434	sdb(1)	cannot handle "c" global variables	bl81-28922
1435	sdb(1)	sdb gave coredump when executed.og=st	bl82-03502
1436	sdiff	enhance sdiff so that exit code is consistent with diff.	bl82-00506
1437	section 3 manual pages	section 3 manual pages should be overhauled	bl82-06911
1438	segf	segf should have same aliases as	bl81-29209

NR	AREA	SUMMARY	MR
		config	
1439	sem	undo structures are not properly deallocated on exit.	ua82-02803
1440	sema	invalid return from lock	bl81-05613
1441	sema	syscall misnamed in sema man page	bl81-05701
1442	sema header files	potential ambiguity	bl81-05427
1443	sema(2)	return code ambiguity	bl81-04110
1444	semaphores	semaphores dont work	bl81-04203
1445	send man page	manual entry for send describing hexadecimal replacement is incorrect.	bl81-18102
1446	send(1c) man page	send(1c) man page must be modified to omit all references to univac	ua81-32303
1447	sentry	sentry does not use the correct umask when doing ;w file	bl82-21102
1448	setjmp(3c)	longjmp should not return the value 0!	bl82-09208
1449	setjmp(3c)	longjmp should not return the value 0!	bl82-09402
1450	setjmp/longjmp (c library)	bobbles register variables in intra-function call.	bl81-26704
1451	setmnt	changes needed for setmnt to work under unix-rt.	bl81-14613
1452	setmnt	compilation of setmnt gives "nmount redefined" message.	bl81-35211
1453	setting up doc	"setting up" documentation does not indicate the use of optimal gap and block settings.	bl81-30012
1454	setting up unix	/dev/swap description too vague.	bl82-01106
1455	setting up unix	clarification needed on obtaining major number for dzb lines	bl81-35703
1456	setting up unix document	all references to dqsrje must be deleted from setting up unix document.	ua81-31701
1457	sgs	complementing a declared short in some constructs will fail.	bl82-06811
1458	sgs	format changes to man pages for libld	bl81-29645
1459	sgs	in cc, pushing an indirect, double on stack doesn't work	bl82-14704
1460	sgs	performance of sgs need to be more competative with vax	bl81-18301
1461	sgs	sgs makefile needs chmods,chowns, chgrps	bl81-22703
1462	sh	cd with null argument defaults strangely.	bl80-33814
1463	sh	sh sets return code 3 on interupt even though sig is caught & prog.	at80-12903
1464	sh	sh will not compile on the pdp11	bl82-12610

NR	AREA	SUMMARY	MR
		in 5.0.	
1465	sh	shell does not diagnose unmatched quotation marks	bl80-21202
1466	sh	shell sets high order bit on characters.	bl79-11406
1467	sh	the sh must be recompiled	bl80-32082
1468	sh	the shell always acts as if it is non-interactive using -e option.	bl80-18203
1469	sh	trap on signal 1 (hangup) doesn't work properly	bl80-32085
1470	sh man page	sh man page doesn't indicate special handling of test with -e option.	bl80-18205
1471	sh mstall	./:mkcmd sh results in 775 bin other.	bl80-31704
1472	sh(1)	entry has ambiguities regarding commands which must begin lines.	bl81-31016
1473	sh(1)	shift should support an argument	bl79-25010
1474	sh(1)	the -t option of test(1) fails to recognize redirection of sh(1) io.	bl80-35107
1475	sh(1) and cd(1)	man entries out of date.	bl81-20309
1476	shared memory	install maus shared memory on pdp11/70	bl80-25412
1477	shared memory	shm.c should be linted and cleaned up.	bl81-30910
1478	shared memory	shmat can panic the system if a starting address is given.	bl81-30909
1479	shared memory	shmdt could crash the system.	ua81-32701
1480	shell	${var=string} assignment doesn't work like var=string	bl80-11411
1481	shell	error in shell parameter substitution within "echo".	bl80-21801
1482	shell	execution of expr script generates memory fault.	bl80-24105
1483	shell	shell forgets to close file descriptors.	bl80-08105
1484	shell	shell parameter subsitution.	bl80-27008
1485	shell(test.c)	null pointer to location 0 dumps core for test	bl82-01331
1486	shm	a shared memory bug can cause user processes to dump core.	bl82-29904
1487	shmem	using sbrk with shared memory may cause the system to lose memory.	bl82-12606
1488	signal()	exit() inside an interrupt handler always returns a 3.	bl79-17103
1489	signal(2)	returns no character count if interrupted	bl82-23111
1490	signal(2)	signal sigcld is reset when caught	bl81-27116
1491	signal(s)	error in documentation of signal(s).	bl80-29503

NR	AREA	SUMMARY	MR
1492	size	size command indicates file than cannot be opened is 'not found'.	bl81-18106
1493	size	size says "not found" when it means "cannot open"	bl81-14212
1494	size of file	when not many changes often starts segmenting over and over	bl80-19801
1495	sizeof	sizeof can give wrong result	bl82-04701
1496	sleep/sched	the way unix handles sleeps causes strange scheduling behavior.	bl81-13516
1497	snoop.command	changes must be made to the structure "event" to allow for a time stamp that has been adde	bl81-29437
1498	snoop command for monitoring vpm devices	snoop only works for device vpm0	bl82-08802
1499	snoop program	snoop must be modified to output msgs to describe newly-added event record types	bl82-03301
1500	snoop.c	snoop and rjeload must be modified to run compatibly with release 5.0 csi	bl82-07007
1501	some manual pages	fix bug in common string definition	bl81-24021
1502	sort	wrong printf	bl80-20507
1503	sort(1)	sort can destroy input file	bl79-17305
1504	space.h	not enough device structures allocated for dmk driver	bl82-16204
1505	spell	"spell +localfile" option does not work	bl82-18811
1506	spell	need capability to update local list	bl79-16607
1507	spell	spell prints things from /usr/lib/tmac/tmac* as misspellings.	bl80-27016
1508	spell	there is a new spell	bl81-22317
1509	split	can not use full path name for target file and	up82-10601
1510	split	split creates unlimited number of files, non-printable names.	bl80-29805
1511	split	split should consider the simple file name.	pt81-28122
1512	split.c	incorrect abort: "more than 12 char...."	bl81-24329
1513	sprintf	sprintf mistakenly tries to write to non-existent device	bl82-07003
1514	src.mk file	/usr/src/cmd/rje/src/src.mk must be modified to install correct rjeload script	ua81-31501
1515	sscanf	sscanf does not fail properly	bl80-25226
1516	sscanf	sscanf returns eof in situations	ua82-07602

NR	AREA	SUMMARY	MR
		where no error should occur.	
1517	sscanf	sscanf thinks centsign is a space.	bl80-33901
1518	sscanf()	sscanf() handles %[	bl80-29808
1519	sscanf, scanf, fscanf	incorrectly handles "[" type scan	bl80-27018
1520	ssignal	ssignal/gsignal will not work for sig#<4.	bl80-33802
1521	stand-alone i/o	upgrade to use rm05 disks.	bl81-05414
1522	standalone	can't boot from rm80	ua82-13001
1523	standalone i/o	sash dose not read files properly on the 11/70.	bl81-05416
1524	standalone library	stand h file system type defined wrong.	ua82-10901
1525	standalone library	standalone library does not support 5.0 sgs header format.	ua82-06104
1526	standalone makefile	the standalone makefile blows up in the sash portion	bl82-13108
1527	start.s	can't boot vax 11/750 with new memory.	cc82-30301
1528	stat	pi in pie.c has only 6 significant figures	bl81-28401
1529	stat routine "gamma"	multiple definition for "signgam" in /usr/src/cmd/graf/src/stat.d/gamma .c	bl81-19511
1530	stat routines	multiply defined symbols when linking programs in /usr/src/cmd/graf/src/stat.d	bl81-19409
1531	stat routines	type mismatch in getline.c, directory /usr/sre/cmd/graf/src/stat.d/getli ne.c	bl81-21611
1532	stat routines	warnings when compiling dist.c and pie.c.	bl81-19512
1533	stdio	1st fprints and fputs to unbuffered stream do not work.	bl81-11203
1534	stdio	fclose sets errno to 2 even if no error.	bl81-34809
1535	stdio	fflush should flush input data	bl81-34912
1536	stdio	fread and fwrite fail in spell programs.	bl82-14809
1537	stdio	stdio.h does not declare type for tmpnam()	bl81-05819
1538	stdio library	fopen(3) successfully opens the a null filename	bl81-08916
1539	stdio(3s)	signals don't seem to work properly in one case.	bl82-01801
1540	stdio(longjmp)	longjmp from an interrupt catching routine fouls buffering.	bl81-14844
1541	stdio/doscan.c	single charater format doesn't	bl81-12105

NR	AREA	SUMMARY	MR
		work.	
1542	strcmp, strncmp	strncmp does not work correctly on vax	bl81-05704
1543	string	strncmp() and strncpy() are inefficient on the vax.	bl81-03707
1544	strip	fix strips archive infinite loo	bl81-30303
1545	strip	fix strips problem with archives	bl81-30010
1546	strip(1)	strip -l causes the message "bus error--core dumped".	bl82-22908
1547	strip.c	change #ifdef vax to include u320.	bl81-30122
1548	stscr.r	need dsrwait	bl82-07719
1549	stscr.r	script handles excessive wak's incorrectly	bl82-02603
1550	stty	please allow the command "stty saner"	bl80-33824
1551	[illegible]	su security change	bl80-32025
1552	su command	su command hangs when system console cannot be written to	bl80-34505
1553	suid	process user id not set	bl81-02812
1554	sum	wrong printf	bl80-20508
1555	super user process locking	super user is allowed only one process to lock.	bl81-03701
1556	sw scrubbing	unix system release 5.0 source and manual must be scrubbed.	bl82-26401
1557	sys_errlist	sys_errlist does not have the new ipc error messages.	bl81-21607
1558	sysdef	sysdef does not recognize 5.0 sgs header format or namelist structure.	ua82-06103
1559	sysdef	sysdef fails with larger kernel text change.	bl80-24802
1560	sysfix	sysfix should be documented	bl80-29103
1561	sysin data sets	sysin data sets are not erased after job comletion	bl80-31224
1562	syslib(sysprd)	problems when rebuilding syslib(sysprd)	bl80-31241
1563	system	provide support for vax-11/750	bl81-24009
1564	system activity package	sar, sag, sadp, and timex won't run	ua82-07101
1565	system activity package	should use ctod for click conversion in sadc.c.	bl81-07608
1566	system activity package	timex prints arbitrary character as the second digitin the time field	bl81-31008
1567	system activity package	upgrade s. a. package to handle general disk driver.	bl81-03510
1568	system activity package	upgrade unix system activity package (vax) to report disk activity for general disk driver	bl81-03509
1569	system activity	upgrade system activity package to	bl81-10410

NR	AREA	SUMMARY	MR
	system	report i/o activities of tu78.	
1570	system configuration file	system configuration file (/etc/system) format change	bl81-35502
1571	system signals	minor signal handling optimization	bl81-04316
1572	sysucs dataset	sysucs dataset for print train info is not documented	bl80-31244
1573	tail	changes needed to tail for portability to unix/370	bl80-14803
1574	tail	tail -12 -f filename does not work.	bl80-25605
1575	tail(1)	tricky usage message	bl81-05119
1576	tan()	tan() function missing from manpage of the manual.	bl81-34313
1577	tbl nroff	spacing around a table.	bl80-23217
1578	td	td does not fit circles on the tek 4014 screen correctly.	bl79-29605
1579	term	renaming job on t4000a half done.	bl81-05114
1580	termlog	termlog file created by a cassi command is not owned by cmts.	bl81-14106
1581	test	test -t default case gives incorrect results	bl82-20408
1582	test	test doesn't know about fifos.	bl80-34901
1583	test	with standard error to file, test -t 2 returns true.	bl80-16109
1584	test command	test command with -t option always returns 0 (true) with argument specified.	bl81-21001
1585	test command	the test command with -c returns wrong value.	bl81-14007
1586	test(1)	undocumented option -p	bl81-08802
1587	text	abstract macro problem	bl81-04327
1588	text	line drawing function with rooten does not work.	bl80-32062
1589	text	make mm, man, and mmt easier to use.	bl80-27717
1590	text	make proc. should remove ft?? files.	bl80-35807
1591	text	need capabilities of reform (1).	bl80-26904
1592	text	new eqn/neqn, tbl, cw	bl80-32029
1593	text	what is font xm?	bl80-22301
1594	text macros	the strings that produce trademarks and	bl80-16804
1595	text, mm	update the programs - cover letter for technical memorandum.	ny82-26408
1596	text.c	text.c should not preallocate swap space for shared text	bl80-32081
1597	text/mm	title interference with af macro.	bl80-22516
1598	text/nroff	nx request does not work.	bl80-32059
1599	the /etc/system	changed the 'alias' names in the	bl82-14710

NR	AREA	SUMMARY	MR
	file	'master' file	
1600	the c compiler, cc	struct can't be initialized to pointer to void function	bl81-27113
1601	the c compiler, cc	struct can't be initialized to pointer to void function	bl81-28935
1602	time(1)	inconsistent use of signal(2) in /usr/src/cmd/time.c.	bl80-17206
1603	times(2)	the synopsis section of times(2) is out of date.	bl81-05609
1604	timex	timex -p on 11/70 fails to list processes	ua82-04701
1605	timex	timex dumps core sometimes	bl80-25506
1606	timex	timex prints canch incorrectly	we81-33713
1607	timex (1)	timex (1) does not report device activity.	bl81-22632
1608	timex(1)	timex -o reports incorrect data.	bl82-15305
1609	timex.c	doesn't handle 2 disk controllers.	bl81-19704
1610	tmac	missing/extra file	bl81-05112
1611	toc routines	warnings when compiling /usr/src/cmd/graf/src/toc.d/vtoc.d /getentry.c	bl81-19508
1612	touch	changes needed to touch for portability to unix/370	bl80-14802
1613	touch	touch usage	bl80-32007
1614	tpr	/usr/src/cmd/lpr/prt/1200.c garbage output.	bl82-01304
1615	tr	tr command exit codes are reversed.	bl78-07301
1616	trace	need new trace driver.	bl81-30148
1617	troff	troff doesn't know how high some characters are	bl80-13601
1618	trs	request for tr reports	bl81-35215
1619	true.1	example in document is bogus.	ua81-31702
1620	ts.c	ts11 driver does not handle short read of long record properly.	ua81-32102
1621	ts11 driver	add ts11 driver to stand-alone i/o library.	bl81-22319
1622	tto.c	printing hangs occasionally when stty = cr2 tab3 bs1.	bl80-34512
1623	tty	raw i/o performance at 300 baud	bl82-01416
1624	tty driver	212a test mode kill unix	bl82-16901
1625	tty driver	ineffective ttread	bl80-19804
1626	tty driver	unix does not support 212a dataphones	bl82-18806
1627	tty line discipline code	tty line discipline should inform operator about no cblocks	up82-02609
1628	tty.c	non-blocking and timed reads don't work	pt80-31811
1629	u-guide	cannot use capitals	bl77-24305
1630	ub	ubminit should allocate only	bl81-07701

NR	AREA	SUMMARY	MR
		registers needed.	
1631	ubmalloc	ubmalloc() lowers processor priority too much	bl81-18307
1632	ulimit()	ulimit(3,0l) gave wrong maximum break value by 4k	bl81-32939
1633	umount	umount should give a more detailed message when unable to unmount.	bl80-30806
1634	uname	"a" and "m" omitted from usage diagnostic.	bl81-02009
1635	uname	networking software should use the "nodename" field of the utsname structure.	bl81-28921
1636	uname	see below.	bl80-13605
1637	uname	uname -n returns the wrong name	bl81-06003
1638	uname	uname should print 9 character fields.	bl80-32418
1639	uname	uname(1) return code	bl81-04338
1640	uname	usage message	bl80-20408
1641	uname and cref	return codes of uname and cref commands	bl81-04311
1642	uname(1)	uname (1) has a bad exit status.	bl81-07507
1643	units	archaic unit centigrade	bl80-26809
1644	units(1)	units prints garbage	bl80-13703
1645	unittab	/usr/lib/unittab is missing	bl82-30806
1646	unix	changes required in source code for portability purposes.	bl80-30804
1647	unix	doesn't support use of 4 megabytes of memory.	bl80-22106
1648	unix	error number out of place	bl80-34515
1649	unix	fill out the library supplied ??search routine sequence	bl81-04002
1650	unix	implement mount table indirection	mh82-05506
1651	unix	install new header scheme	bl80-25419
1652	unix	install new memory management on vax	bl80-25418
1653	unix	need "malloc (coremap,...)" facility in unix/vax.	bl80-24803
1654	unix	need additional kernel text space	bl80-20504
1655	unix	need to have more than 65k bytes of kernel text.	bl80-29519
1656	unix	processes entered more than once on runq	ua81-35505
1657	unix	random thoughts on unix crashes.	bl80-21023
1658	unix	setuid programs should exec setuid programs as themselves	bl80-18411
1659	unix	shell parameters passed invalidly	bl82-20407
1660	unix	the kmc should be able to do dma input.	bl80-22818
1661	unix	uname should return machine type	bl80-26709
1662	unix 3.0	/usr/src/head/rje.h is missing.	bl80-23816

NR	AREA	SUMMARY	MR
	distribution		
1663	unix 3.0 manual frexp(3c)	description of modf (under frexp(3c)) is wrong.	bl81-08205
1664	unix 3.0.1	update to 3.0.1	bl80-25410
1665	unix command tr	supply appropiate return codes in the tr command.	bl81-09903
1666	unix for beginners	unix for beginners should reflect mm.	bl81-12605
1667	unix rje user's guide	unix rje user's guide must be modified to properly reference unix user's manual	bl82-02801
1668	unix source environment	header files are misplaced in unix source	bl80-23401
1669	unix user's manual	unix manual error	bl80-35208
1670	unix user's manual - section 3	section 3 manual pages should be overhauled	bl82-06906
1671	unpack	changes needed to unpack for portability to unix/370	bl80-14801
1672	unpack	does not work for more than 9 files	bl80-21601
1673	unpack(1)	unpack has problems with parity	bl80-26104
1674	unpack, pcat	unpack and pcat can't handle 2.0 packed files on 3.0 vax	bl80-26808
1675	user manual, adm manual	missing commands on "name" line	bl81-20906
1676	user's manual	manual entries for ibm pnch files are misleading.	bl81-22629
1677	utmp.h getlogin.c getut.c ttyslot.c & x.25 files	corrections & updateing of these files for release of 5.0	bl82-09204
1678	uts	bug fixes to tt0.c and tty.c	bl80-26503
1679	uts	install line disciplines (pdp11).	bl80-29516
1680	uts	install line disciplines (vax).	bl80-29517
1681	uts	install new device drivers (pdp11)	bl80-29528
1682	uts	install new device drivers (vax)	bl80-29529
1683	uts	install new messages (vax)	bl80-29530
1684	uts	install new phys. i/o buffer pool (pdp11).	bl80-29514
1685	uts	install new phys. i/o buffer pool (vax).	bl80-29515
1686	uts	install process locking (pdp11).	bl80-29513
1687	uts	install process locking (vax)	bl80-29526
1688	uts	install sleep/nice enhancements (pdp11).	bl80-29527
1689	uts	install sleep/nice enhancements (vax).	bl80-29524

NR	AREA	SUMMARY	MR
1690	uts	install virtual terminal (pdp11).	bl80-29523
1691	uts	install virtual terminal (vax).	bl80-[illegible]9525
1692	uts	install virtual terminal interfaces	bl81-05709
1693	uts/vax/os/sema.c	system may not free semaphores on exits	bl81-10602
1694	uuclean	-d documented as -p	bl81-22517
1695	uuclean	-d does not work	bl81-22519
1696	uuclean	wrong option letter for uuclean directory option on manual page.	bl82-00407
1697	uucp	add option to synopsis	bl80-27716
1698	uucp	bombs talking to systems with long login names	bl81-34909
1699	uucp	comments refer to mhtsa and mhtsb	cc82-28601
1700	uucp	cron can't execute uucp shell programs	bl81-28603
1701	uucp	ecial characters in login sequence done wrong	bl81-03724
1702	uucp	enhancements necessary for oas ii include single job ticket, delete, and determination of	bl82-07708
1703	uucp	entries in logfile for uucp indicate wrong login id.	bl81-04344
1704	uucp	facility for putting comments in l.sys and l.dialcodes file should be documented.	bl81-10308
1705	uucp	incomplete copying	bl80-20308
1706	uucp	incorrect login error message does not appear when wrong passwd used	bl80-33801
1707	uucp	need corrected entry in l.sys.	bl81-05006
1708	uucp	no entry in logfile or stst.uname if uucp login times out	bl81-33807
1709	uucp	security hole in fowarding feature	bl82-14106
1710	uucp	standard uucp system	bl81-01901
1711	uucp	standard uucp system.	bl81-02602
1712	uucp	syslog should be truncated to zero length.	bl81-22908
1713	uucp	use of getopt	bl80-25921
1714	uucp	uucico does not close lines.	bl81-10501
1715	uucp	uucp and mail should use same system prefix convention.	bl81-07202
1716	uucp	uucp does not close dn11 after dialing.	bl81-00605
1717	uucp	uucp does not protect its files from reading everyone	bl81-32320
1718	uucp	uucp entry can cause non-standard shell message in passwd file	bl81-24032
1719	uucp	uucp keeps control of an acu after the dial is complete.	bl82-19714
1720	uucp	uucp l.sys entry	bl80-26903

MRS ANSWERED IN UNIX SYSTEM V

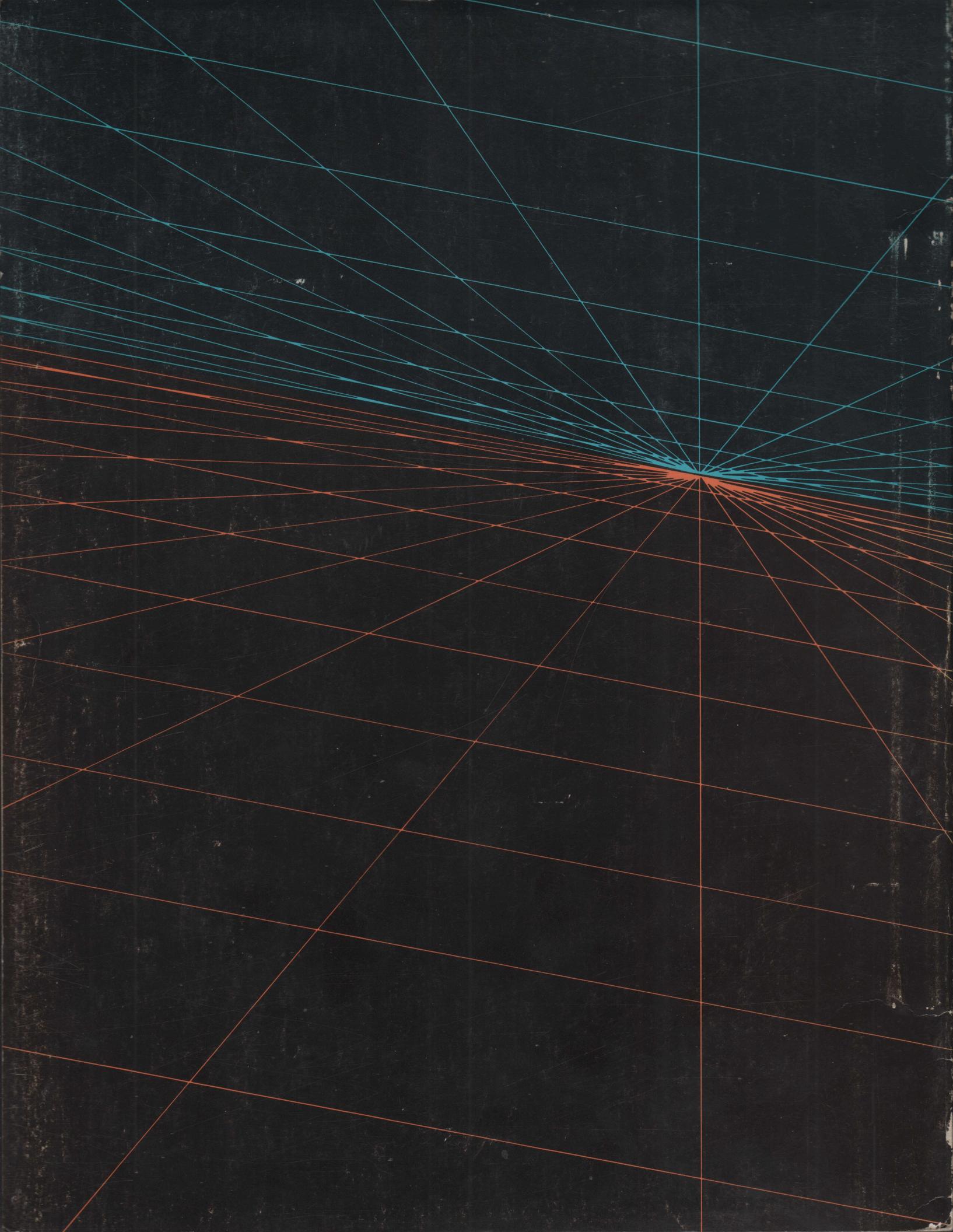
NR	AREA	SUMMARY	MR
1721	uucp	uucp makes wasteful use of dialup lines	bl81-00913
1722	uucp	uucp userfile entry.	bl80-26905
1723	uucp	uucp utilities hang	bl81-04329
1724	uucp	uucping to one's self	bl81-03723
1725	uucp	uuname command run in background terminates on interrupt at terminal.	bl81-10307
1726	uucp	uusub produces core dump when -c option used	bl80-32006
1727	uucp	uuxqt returns garbage signal code in notify msg.	up82-05603
1728	uucp	wrong name in table	cc81-29602
1729	uucp work files	uucp work files 'tm*' not deleted on aborted files transfer	bl80-35501
1730	uudemon.day	path should search in /usr/lib/uucp	bl81-22603
1731	uudemon.hr	path should search in /usr/lib/uucp	bl81-22605
1732	uudemon.wk	path should check in /usr/lib/uucp	bl81-22607
1733	uustat	"-u" option does not accept login names longer than 7 characters	bl81-35702
1734	uustat	octal codes instead of text	bl81-32501
1735	uustat	the uustat command will not recognize the system name which is more than 7 character long.	bl80-33803
1736	uustat	uustat -k does not work	bl81-12807
1737	uustat	uustat man page does not explain status == 3000	bl81-15301
1738	uusub	"uusub -c all" causes overflow when monitoring large "subnetworks"	bl81-28925
1739	uusub	uusub loops when using the -d option	bl80-27410
1740	uusub problems with uudemon.day, uudemon.wk	'uusub' has trouble with released uudemon.* shells	mb81-27803
1741	uusub/uuclean	synopsis should give correct invokation	bl80-26811
1742	uuto/uucp	mail failure goes unreported.	bl80-32608
1743	uuto/uupick	problem with m command of uupick	bl80-25409
1744	uux	uux distributes junk mail	bl81-18802
1745	uux	uux won't pass '!'	bl80-35314
1746	uuxqt	uuxqt inherits some users environment	bl80-35313
1747	vax	documentation out of phase with product.	up82-10406
1748	vax c compiler and optimizer	a problem related to the vax c optimizer	bl81-25413

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
1749	vax c2 optimizer	c2 too pessimistic due to flag being overwritten	bl82-32003
1750	vax optimizer (c2)	vax optimizer mangles 16-bit bit fields on half-word boundaries	bl81-28934
1751	vaxop man page	vaxops(8) incorrect man page.	bl80-25603
1752	vaxops for 11/750 4.2	can't bring up 4.2 On a 11/750 using tu77	uz82-13101
1753	vi	makefile problem make clean and clobber remove the source code for vi.	bl82-19704
1754	vi	the makefile doesn't clean up properly	bl82-17618
1755	vi	vi needs to be delivered for 5.0	bl82-16101
1756	vlx	provide vlx.mk	bl80-33615
1757	volcopy	a timing problem can cause volcopy to sleep forever.	bl80-22406
1758	volcopy	automatic variable "dev" used incorrectly.	bl80-27010
1759	volcopy	file system name	bl81-14208
1760	volcopy	force proper operator response when changing reels.	bl81-07902
1761	volcopy	invalid minor device information on dev error messages.	at79-19815
1762	volcopy	mismatch of volume name of object volume ruins label if object volume is tape	bl81-16408
1763	volcopy	missing "exit (0)" at normal exit	bl81-16405
1764	volcopy	need support for new small block 6250bpi tape drive.	mh82-14002
1765	volcopy	optional argument format is too rigid.	bl81-08202
1766	volcopy	shouldn't look for a null!	bl80-26810
1767	volcopy	six character file system or volume names still do not work.	bl81-20501
1768	volcopy	stronger checks on operator responses.	bl80-23207
1769	volcopy	use of junk file	bl80-25612
1770	volcopy	verification of proper arguments.	bl80-23208
1771	volcopy	volcopy derror when copying to tape	bl81-16404
1772	volcopy	volcopy does not handle a labelled but otherwise blank tape as input	bl81-16703
1773	volcopy	volcopy does not work on multi-reel filesaves.	bl82-21413
1774	volcopy	volcopy doesent accept some command line inputs.	bl81-07904
1775	volcopy	volcopy doesn't work properly when writing out to more than one tape.	bl79-20708
1776	volcopy	volcopy needs changes to support kennedy and cipher tape drives.	mh82-14001

NR	AREA	SUMMARY	MR
1777	volcopy	volcopy needs tape sequence indicator.	bl80-23211
1778	volcopy	volcopy needs to handle 6250 bpi tapes.	bl80-23209
1779	volcopy	volcopy performs incorrect superblock header compare.	we80-29813
1780	volcopy	volcopy scribbles filesystems	bl82-09102
1781	volcopy	volcopy should automatically "label" tapes on writing	bl80-32078
1782	volcopy	volcopy should enforce volname restriction.	bl80-23214
1783	volcopy	volcopy should require confirmation.	bl80-23210
1784	volcopy	when copying tapes, volcopy sometimes goes to sleep forever.	bl80-21101
1785	volcopy(1m)	the - option is undocumented.	bl80-35003
1786	volcopy(1m)	volcopy(1) overwrites super block header.	bl82-13701
1787	volcopy.1m	update volcopy man page	bl81-30808
1788	vp device driver	the vp device driver should use the ioctl commands	bl80-24719
1789	vpm	bisync interpreter performance	bl81-12503
1790	vpm	clearerr should be called in vpmfmt.c after eof is reached.	ua82-02801
1791	vpm	crc16 may be generated inncorrectly	bl82-06002
1792	vpm	develope csi for the vax	ua82-02203
1793	vpm	fix bug in vpm assembler (compiler phase).	bl81-03616
1794	vpm	fix for test program tcmd.c.	bl81-03615
1795	vpm	illegal control character sent in vpm originated transmission.	bl81-28906
1796	vpm	incorrect code picked for release.	bl81-05402
1797	vpm	last write of a transfer does not complete.	bl81-04906
1798	vpm	need csi on 70 to replace vpmb	bl82-06210
1799	vpm	need vpm interpreter fix to allow use of 50 and 56 kb lines	bl81-28904
1800	vpm	the getxbuf primitive tries to transfer 0 bytes.	bl81-30143
1801	vpm	the vpmstart shell doesn't always work.	bl81-30147
1802	vpm	vpm trace facility inadequate	bl80-26511
1803	vpm	vpmfsr has include statements in the wrong order	bl82-13113
1804	vpm	vpmset will not compile	bl82-16517
1805	vpm	vpmwrite crashing the machine.	we81-21111
1806	vpm commands makefile	the vpm commands makefile is wrong.	bl81-30146
1807	vpm makefile	the vpm makefile (vpm.mk) needs to	bl81-30145

NR	AREA	SUMMARY	MR
		build in a different order	
1808	vpm trace driver	time stamp on trace is needed.	bl81-24607
1809	vpm/kmc	restarting rje causes system to hang	bl81-30811
1810	vpmc	bug in xsom primitive	bl81-26006
1811	vpmfmt	new vpmfmt	bl81-30150
1812	vpmfmt	print formats incorrect on trace record. intterm not printed correctly.	ua81-30301
1813	vpmset	vpmset output useless msg when arg. is missing.	bl82-21714
1814	vpmset	vpmset returns garbage.	bl80-33816
1815	vpmset	when vpmset arg missing incorrect error msg output.	bl82-13410
1816	vpmset and vpmfmt commands	bug fixes to vpm commands	bl80-35308
1817	vpmstart	=when a nonexistent device is specified incorrect vpmstart behavior.	bl82-21709
1818	vpmstart	vpmstart accepts illegal magic numbers.	bl82-21712
1819	vpmstart	vpmstart returns an error code of zero after it fails.	bl82-21708
1820	vpmt	the vpmt driver does a character comparison for less than zero.	mh82-15501
1821	vpmt	the vpmt driver must be converted to csi.	bl81-31506
1822	vpmt	vpmt driver should release all buffers on close/clean.	bl81-30151
1823	vpmt	vpmt.c returns a bad value from vpmstat ioctl in some cases	bl82-06905
1824	vtoc	enhance vtoc to draw structure charts as well as hierarchy charts	ms81-24320
1825	write	time stamp on write header	bl81-13409
1826	write	write command from background produces extraneous characters in heading.	bl81-29627
1827	write	write to pipe of more than pipsiz fails.	bl80-22802
1828	write(1)	write should ensure that the person written to can write back	bl81-28213
1829	wump.c	wump.c not portable.	bl81-27210
1830	yacc	5.0 yacc produces huge object files	bl82-08108
1831	yacc	conditional compilation update	bl80-27413
1832	yacc	update of yacc manpage.	bl82-03208
1833	yacc	yacc does not check if it can't create y.tab.h	bl81-09108
1834	yacc	yacc drops core when it can not open y.tab.c	bl81-28014

MRS ANSWERED IN UNIX SYSTEM V

NR	AREA	SUMMARY	MR
1835	yacc	yacc fails to give a warning on overflow.	bl80-23814
1836	yacc	yacc's shift/reduce error messages come out on std. output	bl81-31005